DS5220 Homework 3 (written solution)

Aditya Singh

TOTAL POINTS

95 / 100

QUESTION 1

Problem 1 20 pts

1.1 7 / 7

✓ **- 0 pts Correct**

- 1 pts incorrect/ minor mistake in plots

- 3 pts incorrect best subset

- 1 pts incorrect/ minor mistakes in coefficients

- 1 pts answers not properly stated

- 2 pts missing plots

- 7 pts no submission

1.2 7 / 7

✓ **- 0 pts Correct**

- 3 pts incorrect forward selection subset

- 7 pts incorrect/ missing answers

- 1 pts minor mistake in subset selection

- 3 pts improper/ uninterpretable answer

1.3 6 / 6

✓ **- 0 pts Correct**

- 1 pts incorrect/ optimal lambda not close to the expected 0.03

- 1 pts coefficients are not close to the original coefficients for the optimal lambda value used.

- 1 pts very high MSE

- 1 pts missing coefficients

- 1 pts missing optimal lambda

- 6 pts missing answer/ notebook not running

- 1 pts use all 10 features while training

- 6 pts no submission

QUESTION 2

Problem 2 25 pts

2.1 2 / 2

✓ **- 0 pts Correct**

- 2 pts Missing solution.

2.2 3 / 3

✓ **- 0 pts Correct**

- 1 pts Cross validation error was found instead of the test error.

- 3 pts Missing solution

2.3 5 / 5

✓ **- 0 pts Correct**

- 2 pts Cross validation not performed with different values of lambda.

- 2 pts Test error not reported.

- 5 pts Missing solution

2.4 5 / 5

✓ **- 0 pts Correct**

- 2 pts Cross validation not performed for different values of lambda.

- 2 pts Plot is not correct.

- 2 pts Number of non-zero coefficients not provided.

- 2 pts Test error not reported.

- 1 pts Number of non-zero coefficients not mentioned.

- 5 pts Missing solution

2.5 5 / 5

✓ **- 0 pts Correct**

- 2 pts Cross validation not performed.

- 5 pts Missing solution

2.6 5 / 5

✓ **- 0 pts Correct**

- 2 pts Lesser number of principal components

could have been chosen.

- 5 pts Missing solution

QUESTION 3

Problem 3 25 pts

3.1 0 / 0

✓ **- 0 pts Correct**

- 0 pts Not Done 3(a): Split the data into a training set and a test set with 80% observations in the training set and 20% observations in the test set.

3.2 5 / 5

✓ **- 0 pts Correct**

- 5 pts Not Done 3(b): Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

- 2 pts Didn't plot the tree properly, and interpret the results.

- 1 pts Test MSE not reported.

3.3 10 / 10

✓ **- 0 pts Correct**

- 10 pts Not Done 3(c): Use cross-validation in order to determine the optimal depth of tree max_depth. Similarly, select the minimum number of samples for a split min_samples_split and for a leaf min_samples_leaf using cross-validation. Does selecting these parameters in the regression tree improve the test MSE? Plot the tree again and compare it to the plot from Step (b)

- 5 pts Cross-validation not done

- 2 pts Comment on Findings not done: Does selecting these parameters in the regression tree improve the test MSE? Plot the tree again and compare it to the plot from Step (b)

- 3 pts Tree plot not visible

3.4 3 / 5

- 0 pts Correct

- 5 pts Not Done 3(d): Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the feature_importances_() attribute in DecisionTreeRegressor() to identify which variables are most important.

✓ **- 2 pts Not stated the feature_importances_() attribute in DecisionTreeRegressor() to identify which variables are most important.**

3.5 5 / 5

✓ **- 0 pts Correct**

- 5 pts Not Done 3(e) : Use random forests to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important. Describe the effect of number of trees m as well as the number of variables considered at each split, on the error rate obtained.

- 2 pts Not or incorrectly described the effect of number of trees m as well as the number of variables considered at each split, on the error rate obtained.

QUESTION 4

Problem 4 20 pts

4.1 5 / 5

✓ **- 0 pts Correct**

- 5 pts Not Done 4(a): Begin by creating a bootstrap sample of size 100 and then select a subset of p columns. Vary the value of p and report the p that results in the lowest cross-validation error. Now train a decision tree classifier on the bootstrap sample by setting the depth to 6.

- 2 pts Incorrect Bootstrap sampling of size 100

- 2 pts Incorrect selection of a subset of p columns. Vary the value of p and report the p that results in the lowest cross-validation error.

- 1 pts Incorrect training of decision tree classifier on the bootstrap sample by setting the depth to 6.

4.2 5 / 5

✓ **- 0 pts Correct**

- **5 pts** Not done 4(b): Repeat the above step to generate T ∈ {1,50,100,150,200,300,400} trees and evaluate your training set. Combine the predictions from all trees and assign the final class based on a majority vote of the predictions of every tree. In the case of ties, assign a class randomly among the ties.
- **2 pts** Incorrect T ∈ {1,50,100,150,200,300,400} trees generation and evaluate on the training set.
- **2 pts** Incorrect aggregation of predictions from all trees and assign the final class based on a majority vote of the predictions of every tree.
- **1 pts** In the case of ties, incorrectly assigned a class among the ties.
- **1 pts** The evaluation metric is not as expected.

4.3 5 / 5

✓ **- 0 pts Correct**

- **5 pts** Not done 4(c): Report the training and test error, F1 score, and AUC by varying T in the range {1,50,100,150,200,300,400}.
- **3 pts** Not trained the model with the given mentioned trees.
- **2 pts** Incorrect/incomplete evaluation metric

4.4 5 / 5

✓ **- 0 pts Correct**

- **5 pts** Not Done 4(d): Use an existing package to train a Random Forest algorithm with 10, 50, and 100 decision trees. Report similar metrics on both the training and testing sets. Report the top features having the most influence on the model.
- **2 pts** Not trained a Random Forest algorithm with 10/50/100 decision trees.
- **2 pts** Incorrectly/incompletely reported metrics on both the training and testing sets.
- **1 pts** Incorrectly reported the top 5 features having the most influence on the model.

QUESTION 5

Problem 5 10 pts

5.1 2 / 5

- **0 pts** Correct

✓ **- 1 pts incorrect classification boundary**

✓ **- 2 pts incorrect output representation**

- **1 pts** in correct formula/calculation.
- **5 pts** no submission

1. For eq1, must be + W1 and for eq 2, + W2
2. o/p' =0
3. incorrect sigmoid function

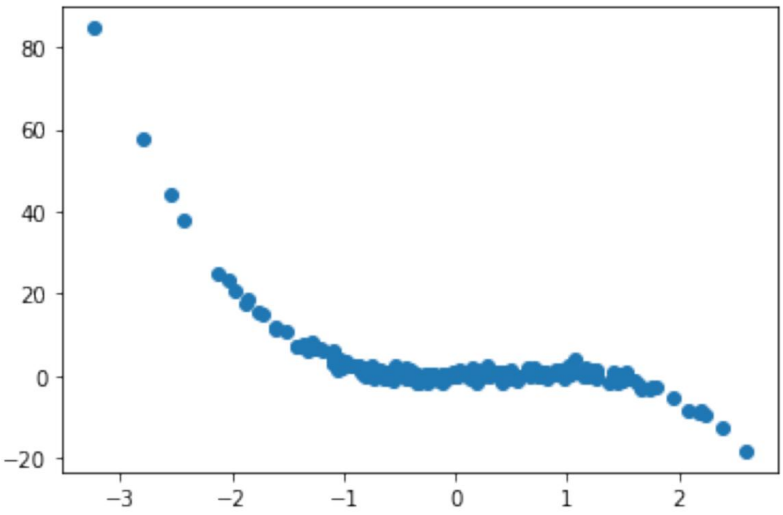
5.2 5 / 5

✓ **- 0 pts Correct**

- **2 pts** missing neural network diagram
- **2 pts** incorrect explaination
- **1 pts** vague /mistake in explanation
- **1 pts** mistake in weights
- **2 pts** no explaination mentioned
- **1 pts** mistake in diagram
- **5 pts** no submission

```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
import statsmodels.api as sm
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

np.random.seed(123)

x = np.random.normal(0,1,(200))
y = x + 2 * x**2 - 2 * x**3 + np.random.normal(0,1,(200))



plt.scatter(x,y)
```

```
<matplotlib.collections.PathCollection at 0x7f151e9e4950>
```




```
def Combinations(L):
    if len(L) == 0:
        return [[]]
    else:
        base = Combinations(L[:-1])
        operator = L[-1:]
        return base + [(b + operator) for b in base]



n_degree = 10
X_columns = ['x' + str(i) for i in range(1,n_degree+1)]
```

```
X_columns
```

```
['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10']
```

```
X = pd.DataFrame()
```

```
i = 1
for c in X_columns:
    X[c] = x**i
    i +=1
```

```
X.head()
```

	x1	x2	x3	x4	x5	x6	x7	x8
0	-1.085631	1.178594	-1.279518	1.389083	-1.508031	1.637165	-1.777356	1.92955
1	0.997345	0.994698	0.992057	0.989424	0.986798	0.984178	0.981565	0.97896
2	0.282978	0.080077	0.022660	0.006412	0.001815	0.000513	0.000145	0.00004
3	-1.506295	2.268924	-3.417668	5.148015	-7.754428	11.680454	-17.594206	26.50205
4	-0.578600	0.334778	-0.193703	0.112076	-0.064847	0.037521	-0.021710	0.01256

```
features = Combinations(X_columns)
features.remove([])
```

```
results = pd.DataFrame()
lst_aic = []
lst_bic = []
lst_rsadj = []
lst_models = []
for m in features:
    x = sm.add_constant(X[m])
    model = sm.OLS(y,X[m]).fit()
    lst_aic.append(model.aic)
    lst_bic.append(model.bic)
    lst_rsadj.append(model.rsquared_adj)
    lst_models.append(m)
```

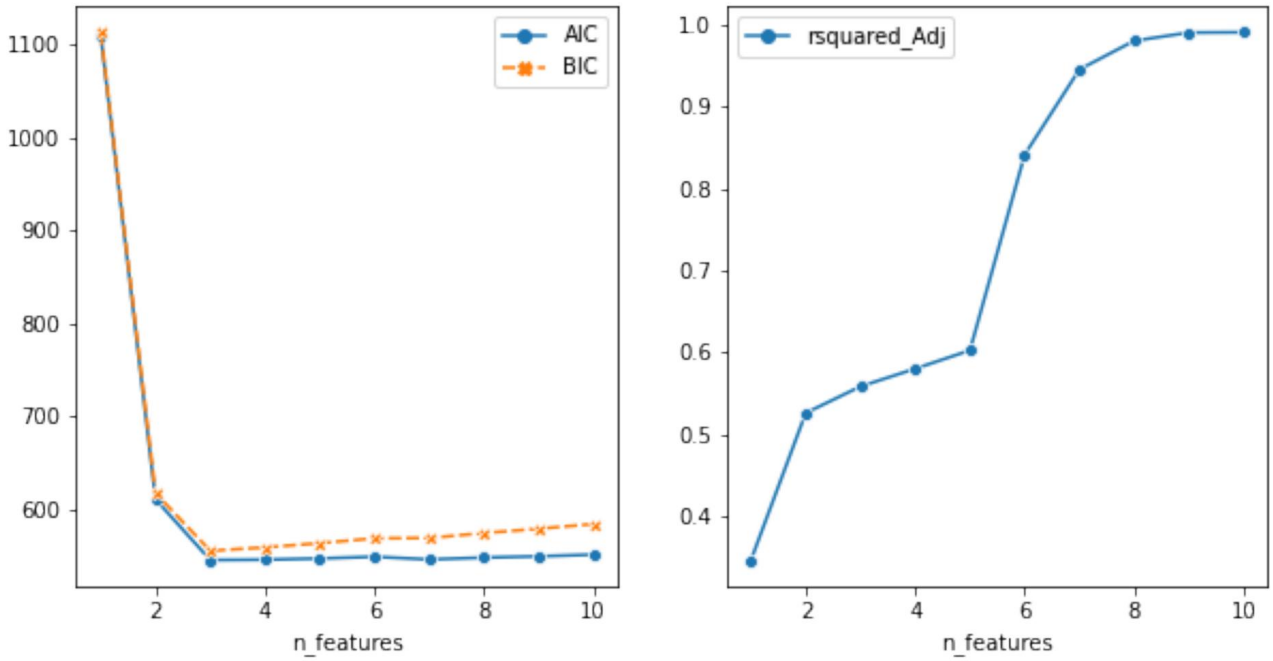
```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: Fut
  x = pd.concat(x[::order], 1)
```

```
results['AIC'] = lst_aic
results['BIC'] = lst_bic
results['rsquared_Adj'] = lst_rsadj
results['Features'] = lst_models

results['n_features'] = results.Features.apply(len)
results_K = results.groupby('n_features').min()
results
```

	AIC	BIC	rsquared_Adj	Features	n_features
0	1374.835640	1378.133957	0.396101	[x1]	1
1	1390.629332	1393.927649	0.346479	[x2]	1
2	1227.095069	1233.691703	0.712925	[x1, x2]	2
3	1127.209016	1130.507333	0.824915	[x3]	1
4	1103.057379	1109.654014	0.845599	[x1, x3]	2
...	...	...	...	...	...
1018	556.906333	586.591189	0.990274	[x1, x2, x4, x5, x6, x7, x8, x9, x10]	9
1019	559.914519	586.301058	0.990079	[x3, x4, x5, x6, x7, x8, x9, x10]	8
1020	558.295588	587.980444	0.990206	[x1, x3, x4, x5, x6, x7, x8, x9, x10]	9
1021	555.592286	585.277143	0.990338	[x2, x3, x4, x5, x6, x7, x8, x9, x10]	9

```
fig, axes = plt.subplots(1, 2,figsize = (10,5))
sns.lineplot(ax = axes[0],data = results_K[['AIC','BIC']],markers=True)
sns.lineplot(ax = axes[1],data = results_K[['rsquared_Adj']],markers=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f151d8dbe50>
```




```
model_aic = results[results.AIC == results.AIC.min()]
model_bic = results[results.BIC == results.BIC.min()]
model_adj_rsq = results[results.rsquared_Adj == results.rsquared_Adj.max()]
```

```
print(model_aic)
print(model_bic)
print(model_adj_rsq)
```

```
           AIC         BIC  rsquared_Adj      Features  n_features
6  545.216313  555.111266      0.990556  [x1, x2, x3]           3
           AIC         BIC  rsquared_Adj      Features  n_features
6  545.216313  555.111266      0.990556  [x1, x2, x3]           3
            AIC         BIC  rsquared_Adj                        Features  \
686  545.863439  568.951661      0.990708  [x1, x2, x3, x4, x6, x8, x10]

     n_features
686           7
```

From the above deduction we can see that at number of feature = 3 we get the best values

PART B

1.1 7 / 7

✓ **- 0 pts** **Correct**

- 1 pts incorrect/ minor mistake in plots

- 3 pts incorrect best subset

- 1 pts incorrect/ minor mistakes in coefficients

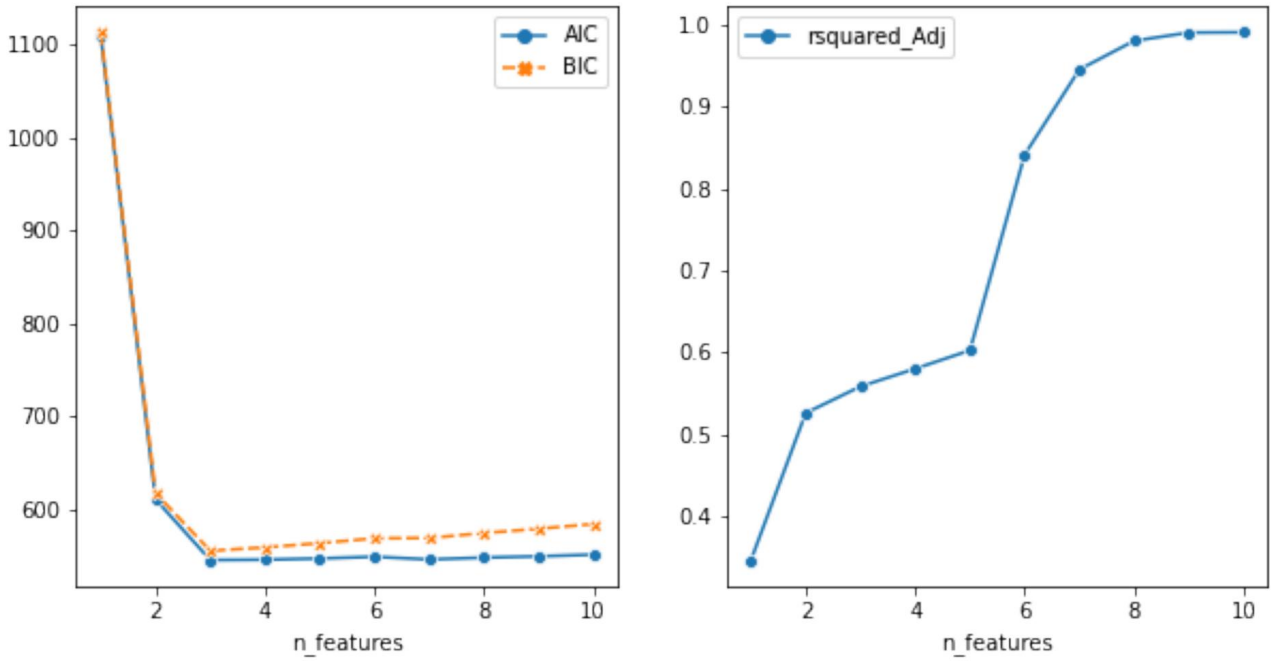
- 1 pts answers not properly stated

- 2 pts missing plots

- 7 pts no submission

```
fig, axes = plt.subplots(1, 2,figsize = (10,5))
sns.lineplot(ax = axes[0],data = results_K[['AIC','BIC']],markers=True)
sns.lineplot(ax = axes[1],data = results_K[['rsquared_Adj']],markers=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f151d8dbe50>
```




```
model_aic = results[results.AIC == results.AIC.min()]
model_bic = results[results.BIC == results.BIC.min()]
model_adj_rsq = results[results.rsquared_Adj == results.rsquared_Adj.max()]
```

```
print(model_aic)
print(model_bic)
print(model_adj_rsq)
```

```
           AIC         BIC  rsquared_Adj      Features  n_features
6  545.216313  555.111266      0.990556  [x1, x2, x3]           3
           AIC         BIC  rsquared_Adj      Features  n_features
6  545.216313  555.111266      0.990556  [x1, x2, x3]           3
            AIC         BIC  rsquared_Adj                        Features  \
686  545.863439  568.951661      0.990708  [x1, x2, x3, x4, x6, x8, x10]

     n_features
686           7
```

From the above deduction we can see that at number of feature = 3 we get the best values

PART B

```
p = X_columns[:]
model_i =[]
step_aic = []
step_models = []
for i in range(0,len(p)):
    x = model_i + [p[0]]
    x=sm.add_constant(X[x])
    model_0 =sm.OLS(y,x).fit()
    min_aic = model_0.aic
    p_i = p[0]
    for f in p:
        x = model_i + [f]
        x = sm.add_constant(X[x])
        model = sm.OLS(y,x).fit()
        if model.aic < min_aic:
            min_aic = model.aic
            p_i = f
    model_i += [p_i]
    step_models.append(model_i.copy())
    step_aic.append(min_aic)
    p.remove(p_i)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tsa/tsatools.py:142: Fut
  x = pd.concat(x[::order], 1)
```

```
step_models
```

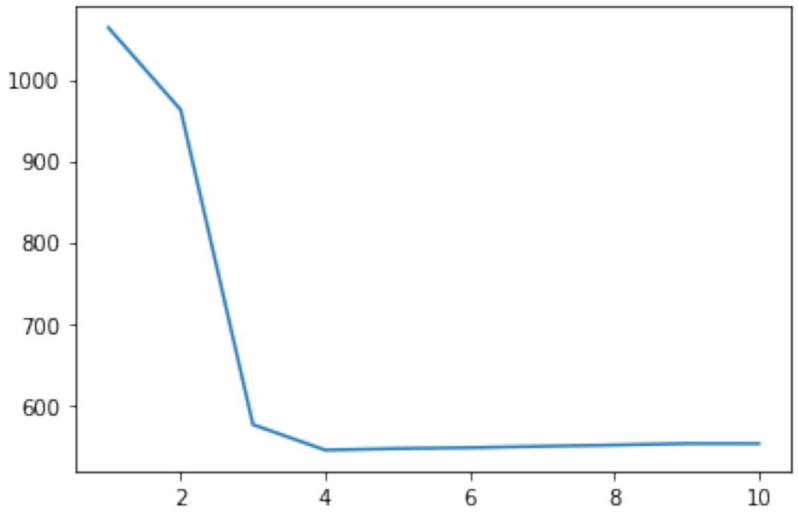
```
[['x5'],
 ['x5', 'x2'],
 ['x5', 'x2', 'x3'],
 ['x5', 'x2', 'x3', 'x1'],
 ['x5', 'x2', 'x3', 'x1', 'x9'],
 ['x5', 'x2', 'x3', 'x1', 'x9', 'x7'],
 ['x5', 'x2', 'x3', 'x1', 'x9', 'x7', 'x10'],
 ['x5', 'x2', 'x3', 'x1', 'x9', 'x7', 'x10', 'x6'],
 ['x5', 'x2', 'x3', 'x1', 'x9', 'x7', 'x10', 'x6', 'x8'],
 ['x5', 'x2', 'x3', 'x1', 'x9', 'x7', 'x10', 'x6', 'x8', 'x4']]
```

```
step_aic
```

```
[1063.8489818852993,
 962.8140676790715,
 575.6144215616509,
 544.2577728896879,
 546.1228238960184,
 547.1866861752989,
 548.9194707797913,
 550.4544109727458,
 552.2405334119981,
 552.1854977568105]
```

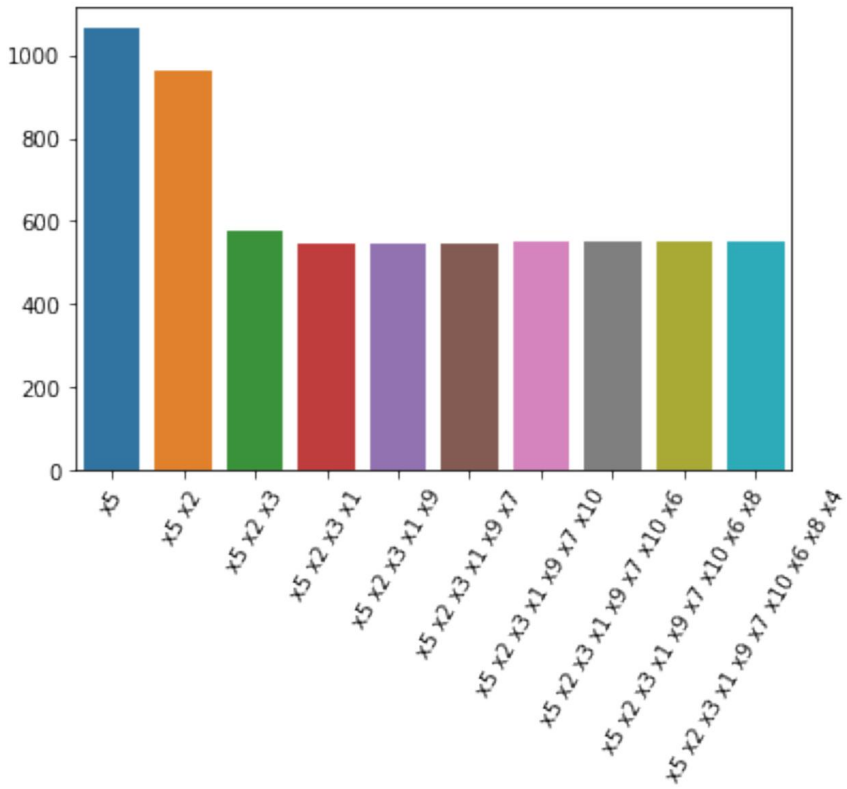
```
sns.lineplot(x = [len(i) for i in step_models],y = step_aic)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f151d4a88d0>
```



```
chart = sns.barplot(x=[' '.join(i) for i in step_models],y = step_aic)
plt.xticks(rotation=60)
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 <a list of 10 Text major ticklabel objects>)
```



At number of features equals 4 we get the lowest AIC however its very close to out previous prediction of 3 features

Part C

1.2 **7 / 7**

✓ **- 0 pts** **Correct**

- 3 pts incorrect forward selection subset

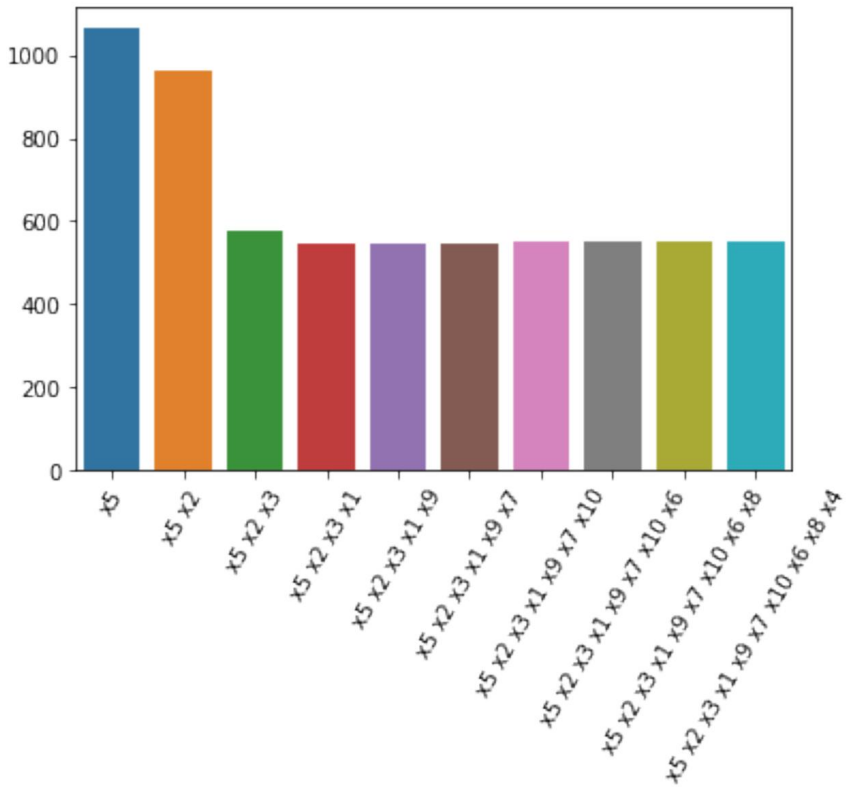
- 7 pts incorrect/ missing answers

- 1 pts minor mistake in subset selection

- 3 pts improper/ uninterpretable answer

```
chart = sns.barplot(x=[' '.join(i) for i in step_models],y = step_aic)
plt.xticks(rotation=60)
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 <a list of 10 Text major ticklabel objects>)
```



At number of features equals 4 we get the lowest AIC however its very close to out previous prediction of 3 features

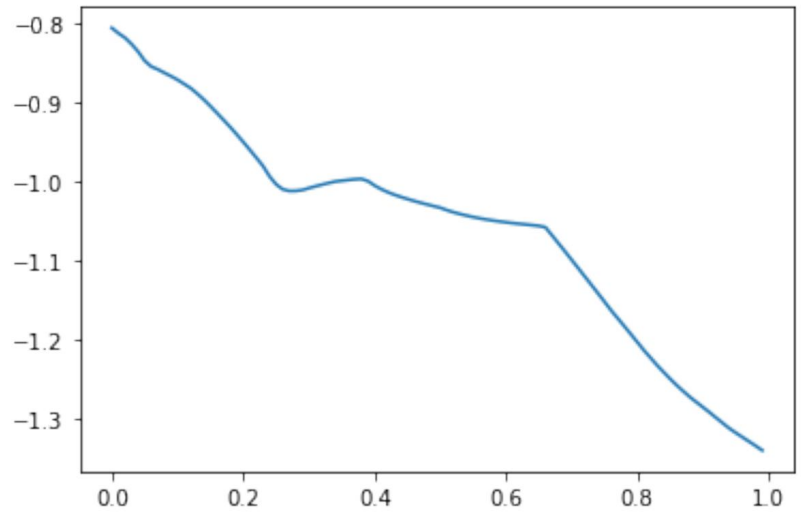
Part C

```
from sklearn.linear_model import LassoCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from numpy import absolute

# define model
lasso_model = Lasso()

# define model evaluation method
cv = KFold(n_splits=10, shuffle = True, random_state=1)

# define grid
grid = dict()
grid['alpha'] = np.arange(0, 1, 0.01)
# define search
search = GridSearchCV(lasso_model, grid, scoring='neg_mean_absolute_error', cv=c

# perform the search
results = search.fit(X, y)
# summarize
print('MAE: %.3f' % results.best_score_)
```

```
MAE: -0.806
```

```
alpha_score = results.cv_results_['mean_test_score']
alphas = np.arange(0, 1, 0.01)
sns.lineplot(alphas, alpha_score)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f151bfce3d0>
```



Above is plot of error v/s lambda values

```
from sklearn.linear_model import LassoCV

# define model
laso_cv = LassoCV(normalize=True, max_iter=10**8,cv=5)


laso_cv.fit(X,y)
# perform the search
results = laso_cv.fit(X,y)
# summarize
print('Alpha: %.3f' % results.alpha_)
print('Config: %s' % results.coef_)
```

```
Alpha: 0.001
Config: [ 0.74580559  2.04739486 -1.88494253  0.         -0.0075694   0.
 -0.          0.          0.         -0.        ]
```

Lasso CV gives lambda as 0.001 with the above mentioned coefficients as we can see some of the redundant features are assigned the value 0 as per Lasso penalization

1.3 6 / 6

✓ **- 0 pts** **Correct**

- 1 pts incorrect/ optimal lambda not close to the expected 0.03

- 1 pts coefficients are not close to the original coefficients for the optimal lambda value used.

- 1 pts very high MSE

- 1 pts missing coefficients

- 1 pts missing optimal lambda

- 6 pts missing answer/ notebook not running

- 1 pts use all 10 features while training

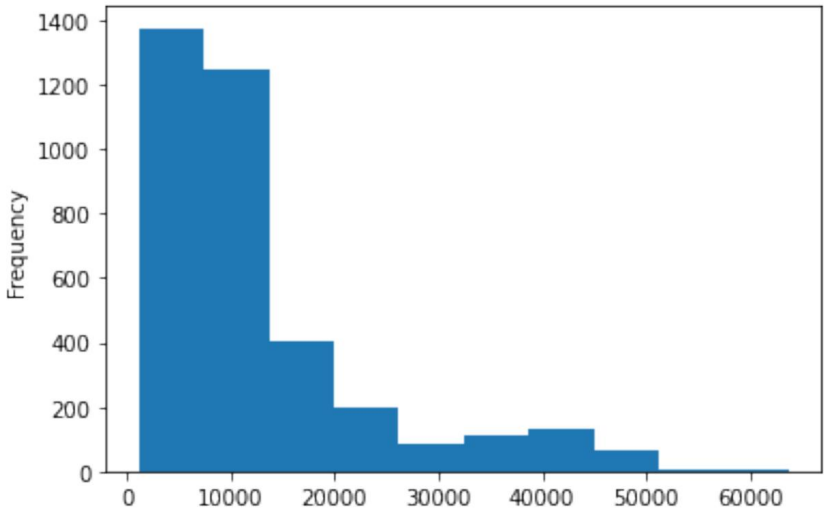
- 6 pts no submission

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import Lasso
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from scipy.stats import norm
from sklearn.linear_model import LassoCV, Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.metrics import make_scorer
from sklearn.cross_decomposition import PLSRegression
```

```
data = pd.read_csv('Train_Data.csv')
```

```
data['charges'].plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc7e46e9d90>
```



```
data.describe()
```

	age	bmi	children	charges
count	3630.000000	3630.000000	3630.000000	3630.000000
mean	38.887036	30.629652	2.503581	12784.808644
std	12.151029	5.441307	1.712568	10746.166743
min	18.000000	15.960000	0.000000	1121.873900
25%	29.000000	26.694526	1.000000	5654.818262
50%	39.170922	30.200000	3.000000	9443.807222
75%	48.343281	34.100000	4.000000	14680.407505
max	64.000000	53.130000	5.000000	63770.428010

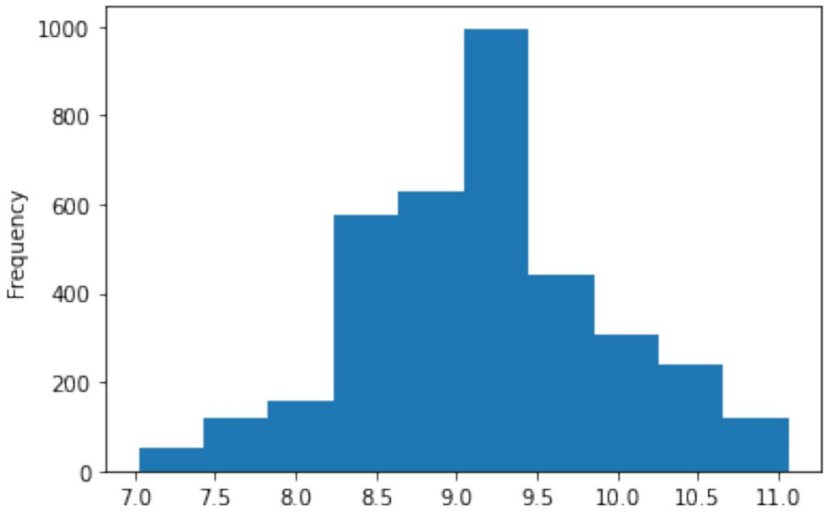
```
# y_bc,lam, ci= boxcox(data['charges'],alpha=0.05)
data['charges'] = np.log(data['charges'])
```

```
data.describe()
```

	age	bmi	children	charges
count	3630.000000	3630.000000	3630.000000	3630.000000
mean	38.887036	30.629652	2.503581	9.164151
std	12.151029	5.441307	1.712568	0.763456
min	18.000000	15.960000	0.000000	7.022756
25%	29.000000	26.694526	1.000000	8.640263
50%	39.170922	30.200000	3.000000	9.153114
75%	48.343281	34.100000	4.000000	9.594268
max	64.000000	53.130000	5.000000	11.063045

```
data['charges'].plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc7e3c30f90>
```




```
data = pd.get_dummies(data, columns = ['sex','smoker','region'],drop_first=True)
```

```
X = data.drop(columns = 'charges')
y = data['charges']
```

A

```
#split the data in 80-20
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2,random_s
```

B: Linear Regression

```
linreg = LinearRegression().fit(X_train,y_train)
```

```
linreg.score(X_train,y_train)
```

```
0.7284617510106679
```

```
linreg.score(X_test,y_test)
```

```
0.705530146117044
```

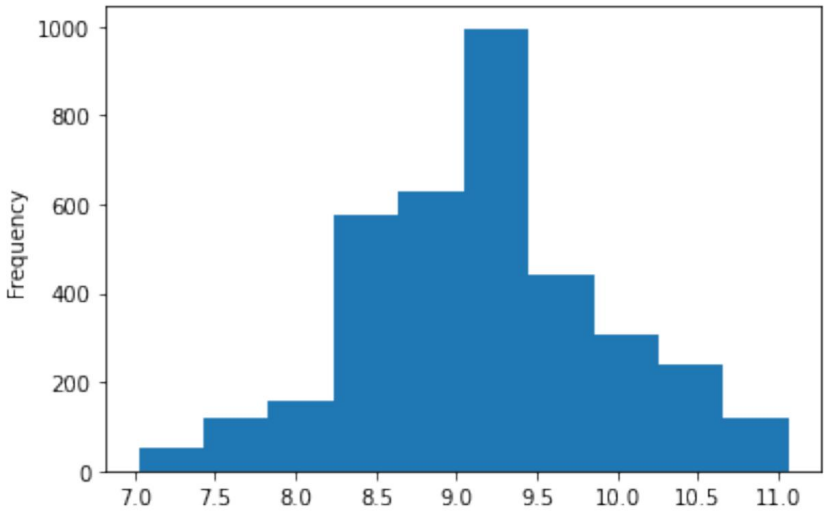
2.1 2 / 2

✓ **- 0 pts** Correct

- 2 pts Missing solution.

```
data['charges'].plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc7e3c30f90>
```




```
data = pd.get_dummies(data, columns = ['sex','smoker','region'],drop_first=True)
```

```
X = data.drop(columns = 'charges')
y = data['charges']
```

A

```
#split the data in 80-20
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2,random_s
```

B: Linear Regression

```
linreg = LinearRegression().fit(X_train,y_train)
```

```
linreg.score(X_train,y_train)
```

```
0.7284617510106679
```

```
linreg.score(X_test,y_test)
```

```
0.705530146117044
```

```
y_pred = linreg.predict(X_test)
print("The RMSE Error {}".format(mean_squared_error(y_true=y_test,y_pred=y_pred
                                                    ,squared=False)))
```

```
The RMSE Error 0.4041331698122962
```

```
print("The coefficients are {} and intercept is {}".format(linreg.coef_,
                                                           linreg.intercept_))
```

```
The coefficients are [ 0.03098396  0.00940595  0.10629314  0.02247008  1.41
 -0.24342369 -0.24360573] and intercept is 7.357833794000509
```

The Error after fitting Linear model with least square is 0.4041

C: Ridge Regression

```
lambdas = 10**np.linspace(10,-2,100)
```

```
ridge = RidgeCV(alphas = lambdas,cv =10, normalize= True )
```

```
ridge.fit(X_train,y_train)
```

```
ridge.alpha_
```

```
0.01
```

```
ridge_ = Ridge(alpha = ridge.alpha_).fit(X_train,y_train)
```

```
ridge_.score(X_train, y_train)
```

```
0.7284617506348426
```

```
ridge.score(X_test, y_test)
```

```
0.7058666647467422
```

```
pred = ridge.predict(X_test)
```

2.2 3 / 3

✓ **- 0 pts** **Correct**

- 1 pts Cross validation error was found instead of the test error.

- 3 pts Missing solution

```
y_pred = linreg.predict(X_test)
print("The RMSE Error {}".format(mean_squared_error(y_true=y_test,y_pred=y_pred
                                                    ,squared=False)))
```

```
The RMSE Error 0.4041331698122962
```

```
print("The coefficients are {} and intercept is {}".format(linreg.coef_,
                                                           linreg.intercept_))
```

```
The coefficients are [ 0.03098396  0.00940595  0.10629314  0.02247008  1.41
 -0.24342369 -0.24360573] and intercept is 7.357833794000509
```

The Error after fitting Linear model with least square is 0.4041

C: Ridge Regression

```
lambdas = 10**np.linspace(10,-2,100)
```

```
ridge = RidgeCV(alphas = lambdas,cv =10, normalize= True )
```

```
ridge.fit(X_train,y_train)
```

```
ridge.alpha_
```

```
0.01
```

```
ridge_ = Ridge(alpha = ridge.alpha_).fit(X_train,y_train)
```

```
ridge_.score(X_train, y_train)
```

```
0.7284617506348426
```

```
ridge.score(X_test, y_test)
```

```
0.7058666647467422
```

```
pred = ridge.predict(X_test)
```

```
mean_squared_error(y_true= y_test, y_pred= pred,squared = False)
```

```
0.4039021831501731
```

```
print("The coefficients are {} and intercept is {}".format(ridge.coef_,
                                                           ridge.intercept_))
```

```
The coefficients are [ 0.03066687  0.00942884  0.10433398  0.02355771  1.40
 -0.23457737 -0.23696232] and intercept is 7.370163342797556
```

The error through Ridge model is 0.4039 and Lambda after CV is 0.01

D: Lasso Regression

```
laso_cv = LassoCV(cv=5).fit(X_train,y_train)
```

```
laso_cv.alpha_
```

```
0.004642162765505469
```

```
lasso = Lasso(alpha=laso_cv.alpha_).fit(X_train,y_train)
```

```
print("The RMSE for Lasso model is {}".format(mean_squared_error(y_true = y_test
                           , y_pred = lasso.predict(X_test),squared = False)))
```

```
The RMSE for Lasso model is 0.4042890803579888
```

```
print("The coefficients of Lasso model are {} and the intercept is {}".
      format(lasso.coef_,lasso.intercept_))
```

```
The coefficients of Lasso model are [ 0.03087823  0.00925496  0.10210665  0
 -0.16509569 -0.16474041] and the intercept is 7.327744021280191
```

Error is 0.4042, Lambda through cv is 0.0046. If we ignore 0.009 which will be ignored if we increase the number of iterations we get 7 coefficients

Principal Component Analysis

2.3 5 / 5

✓ **- 0 pts** **Correct**

- 2 pts Cross validation not performed with different values of lambda.

- 2 pts Test error not reported.

- 5 pts Missing solution

```
mean_squared_error(y_true= y_test, y_pred= pred,squared = False)
```

```
0.4039021831501731
```

```
print("The coefficients are {} and intercept is {}".format(ridge.coef_,
                                                           ridge.intercept_))
```

```
The coefficients are [ 0.03066687  0.00942884  0.10433398  0.02355771  1.40
 -0.23457737 -0.23696232] and intercept is 7.370163342797556
```

The error through Ridge model is 0.4039 and Lambda after CV is 0.01

D: Lasso Regression

```
laso_cv = LassoCV(cv=5).fit(X_train,y_train)
```

```
laso_cv.alpha_
```

```
0.004642162765505469
```

```
lasso = Lasso(alpha=laso_cv.alpha_).fit(X_train,y_train)
```

```
print("The RMSE for Lasso model is {}".format(mean_squared_error(y_true = y_test
                           , y_pred = lasso.predict(X_test),squared = False)))
```

```
The RMSE for Lasso model is 0.4042890803579888
```

```
print("The coefficients of Lasso model are {} and the intercept is {}".
      format(lasso.coef_,lasso.intercept_))
```

```
The coefficients of Lasso model are [ 0.03087823  0.00925496  0.10210665  0
 -0.16509569 -0.16474041] and the intercept is 7.327744021280191
```

Error is 0.4042, Lambda through cv is 0.0046. If we ignore 0.009 which will be ignored if we increase the number of iterations we get 7 coefficients

Principal Component Analysis

```
flow = Pipeline([('scaler',StandardScaler()),
                 ('PCA',PCA()),
                 ('reg',LinearRegression())])


mse = make_scorer(mean_squared_error,greater_is_better=False)


cv = GridSearchCV(estimator=flow,
                  scoring = mse,
                  param_grid={'PCA__n_components':[*range(1,len(data.columns))]}


cv.fit(X_train,y_train)
```

```
    GridSearchCV(estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                           ('PCA', PCA()),
                                           ('reg', LinearRegression())]),
                 param_grid={'PCA__n_components': [1, 2, 3, 4, 5, 6, 7, 8]},
                 scoring=make_scorer(mean_squared_error,
    greater_is_better=False))
```

```
print(cv.best_estimator_.get_params())
```

```
    {'memory': None, 'steps': [('scaler', StandardScaler()), ('PCA', PCA(n_comp
```

```
print("The RMSE for PCA+Regression model is {}".format(mean_squared_error(
    y_true = y_test, y_pred = cv.predict(X_test),squared = False)))
```

```
    The RMSE for PCA+Regression model is 0.40413316981229597
```

The Number of components through cross validation is 8 with error of 0.4041

Least Squared Model

```
flow = Pipeline([('scaler',StandardScaler()),
                 ('pls',PLSRegression())])


mse = make_scorer(mean_squared_error,greater_is_better=True)
```

2.4 5 / 5

✓ **- 0 pts** Correct

- 2 pts Cross validation not performed for different values of lambda.

- 2 pts Plot is not correct.

- 2 pts Number of non-zero coefficients not provided.

- 2 pts Test error not reported.

- 1 pts Number of non-zero coefficients not mentioned.

- 5 pts Missing solution

```
mean_squared_error(y_true= y_test, y_pred= pred,squared = False)
```

```
0.4039021831501731
```

```
print("The coefficients are {} and intercept is {}".format(ridge.coef_,
                                                           ridge.intercept_))
```

```
The coefficients are [ 0.03066687  0.00942884  0.10433398  0.02355771  1.40
 -0.23457737 -0.23696232] and intercept is 7.370163342797556
```

The error through Ridge model is 0.4039 and Lambda after CV is 0.01

D: Lasso Regression

```
laso_cv = LassoCV(cv=5).fit(X_train,y_train)
```

```
laso_cv.alpha_
```

```
0.004642162765505469
```

```
lasso = Lasso(alpha=laso_cv.alpha_).fit(X_train,y_train)
```

```
print("The RMSE for Lasso model is {}".format(mean_squared_error(y_true = y_test
                           , y_pred = lasso.predict(X_test),squared = False)))
```

```
The RMSE for Lasso model is 0.4042890803579888
```

```
print("The coefficients of Lasso model are {} and the intercept is {}".
      format(lasso.coef_,lasso.intercept_))
```

```
The coefficients of Lasso model are [ 0.03087823  0.00925496  0.10210665  0
 -0.16509569 -0.16474041] and the intercept is 7.327744021280191
```

Error is 0.4042, Lambda through cv is 0.0046. If we ignore 0.009 which will be ignored if we increase the number of iterations we get 7 coefficients

Principal Component Analysis

```
flow = Pipeline([('scaler',StandardScaler()),
                 ('PCA',PCA()),
                 ('reg',LinearRegression())])


mse = make_scorer(mean_squared_error,greater_is_better=False)


cv = GridSearchCV(estimator=flow,
                  scoring = mse,
                  param_grid={'PCA__n_components':[*range(1,len(data.columns))]}


cv.fit(X_train,y_train)
```

```
    GridSearchCV(estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                           ('PCA', PCA()),
                                           ('reg', LinearRegression())]),
                 param_grid={'PCA__n_components': [1, 2, 3, 4, 5, 6, 7, 8]},
                 scoring=make_scorer(mean_squared_error,
    greater_is_better=False))
```

```
print(cv.best_estimator_.get_params())
```

```
    {'memory': None, 'steps': [('scaler', StandardScaler()), ('PCA', PCA(n_comp
```

```
print("The RMSE for PCA+Regression model is {}".format(mean_squared_error(
    y_true = y_test, y_pred = cv.predict(X_test),squared = False)))
```

```
    The RMSE for PCA+Regression model is 0.40413316981229597
```

The Number of components through cross validation is 8 with error of 0.4041

Least Squared Model

```
flow = Pipeline([('scaler',StandardScaler()),
                 ('pls',PLSRegression())])


mse = make_scorer(mean_squared_error,greater_is_better=True)
```

2.5 **5 / 5**

✓ **- 0 pts** **Correct**

- 2 pts Cross validation not performed.

- 5 pts Missing solution

```
flow = Pipeline([('scaler',StandardScaler()),
                 ('PCA',PCA()),
                 ('reg',LinearRegression())])


mse = make_scorer(mean_squared_error,greater_is_better=False)


cv = GridSearchCV(estimator=flow,
                  scoring = mse,
                  param_grid={'PCA__n_components':[*range(1,len(data.columns))]}


cv.fit(X_train,y_train)
```

```
    GridSearchCV(estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                           ('PCA', PCA()),
                                           ('reg', LinearRegression())]),
                 param_grid={'PCA__n_components': [1, 2, 3, 4, 5, 6, 7, 8]},
                 scoring=make_scorer(mean_squared_error,
    greater_is_better=False))
```

```
print(cv.best_estimator_.get_params())
```

```
    {'memory': None, 'steps': [('scaler', StandardScaler()), ('PCA', PCA(n_comp
```

```
print("The RMSE for PCA+Regression model is {}".format(mean_squared_error(
    y_true = y_test, y_pred = cv.predict(X_test),squared = False)))
```

```
    The RMSE for PCA+Regression model is 0.40413316981229597
```

The Number of components through cross validation is 8 with error of 0.4041

Least Squared Model

```
flow = Pipeline([('scaler',StandardScaler()),
                 ('pls',PLSRegression())])


mse = make_scorer(mean_squared_error,greater_is_better=True)
```

```
cv_ = GridSearchCV(estimator = flow,
                   scoring = mse,
                   param_grid={'pls__n_components':
                         [*range(1,len(data.columns))]}).fit(X_train,y_train)


print(cv_.best_estimator_.get_params())


print("The RMSE for PLS model is {}".format(mean_squared_error(y_true = y_test,
                                y_pred = cv_.predict(X_test),squared = False)))
```

```
The RMSE for PLS model is 0.4462887740792081
```

The number of components throughs PLS is 7 with error or 0.4462

2.6 5 / 5

✓ **- 0 pts** Correct

- 2 pts Lesser number of principal components could have been chosen.

- 5 pts Missing solution

```
from sklear.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
```

```
data = pd.read_csv('College.csv')
```

```
data.head(5)
```

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc	Top25perc	F.Undergrad
0	Abilene Christian University	Yes	1660	1232	721	23	52	2885
1	Adelphi University	Yes	2186	1924	512	16	29	2683
2	Adrian College	Yes	1428	1097	336	22	50	1036
3	Agnes Scott College	Yes	417	349	137	60	89	510
4	Alaska Pacific University	Yes	193	146	55	16	44	249

```
data.columns
```

```
Index(['Unnamed: 0', 'Private', 'Apps', 'Accept', 'Enroll', 'Top10perc',
       'Top25perc', 'F.Undergrad', 'P.Undergrad', 'Outstate',
'Room.Board',
       'Books', 'Personal', 'PhD', 'Terminal', 'S.F.Ratio', 'perc.alumni',
       'Expend', 'Grad.Rate'],
      dtype='object')
```

```
data.rename(columns = {'Unnamed: 0':'Colleges'},inplace=True)

data['accept_rate'] = data['Accept']/data['Apps']

data = pd.get_dummies(data,columns = ['Private'],drop_first=True)

data.drop(columns=['Accept','Apps'],inplace = True)

X = data.drop(columns=['accept_rate','Colleges'])

y=data['accept_rate']
```

A: Train Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state = 1,
                                                    test_size=0.2)
```

B: Decision Tree

```
dec_tree = DecisionTreeRegressor().fit(X_train,y_train)

y_pred = dec_tree.predict(X_test)

print('The MSE error for the Decission Tree Regressor is {}'.
      format(mean_squared_error(y_true = y_test, y_pred=y_pred)))
mse_dec_tree = mean_squared_error(y_true = y_test, y_pred=y_pred)
print(dec_tree.score(X_train,y_train))
```

The MSE error for the Decission Tree Regressor is 0.022141440157720658 1.0

3.1 0 / 0

✓ **- 0 pts** Correct

- 0 pts Not Done 3(a): Split the data into a training set and a test set with 80% observations in the training set and 20% observations in the test set.

```
data.rename(columns = {'Unnamed: 0':'Colleges'},inplace=True)

data['accept_rate'] = data['Accept']/data['Apps']

data = pd.get_dummies(data,columns = ['Private'],drop_first=True)

data.drop(columns=['Accept','Apps'],inplace = True)

X = data.drop(columns=['accept_rate','Colleges'])

y=data['accept_rate']
```

A: Train Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state = 1,
                                                    test_size=0.2)
```

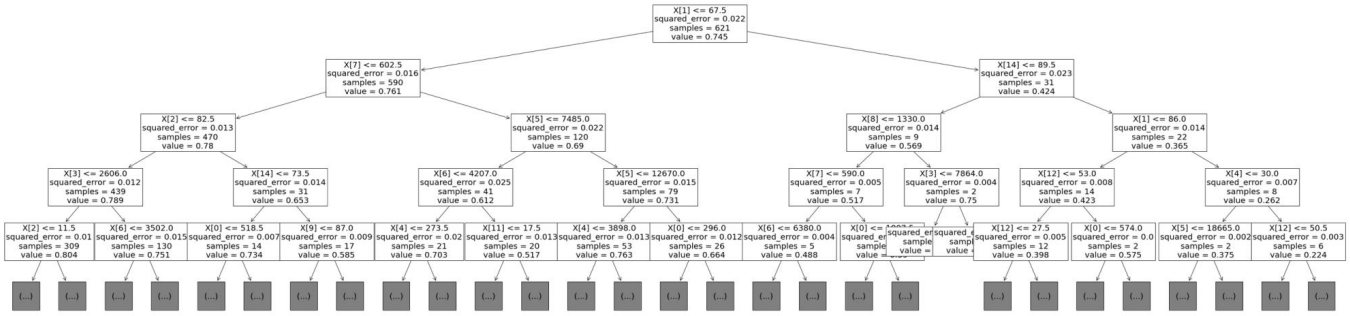
B: Decision Tree

```
dec_tree = DecisionTreeRegressor().fit(X_train,y_train)

y_pred = dec_tree.predict(X_test)

print('The MSE error for the Decission Tree Regressor is {}'.
      format(mean_squared_error(y_true = y_test, y_pred=y_pred)))
mse_dec_tree = mean_squared_error(y_true = y_test, y_pred=y_pred)
print(dec_tree.score(X_train,y_train))
```

The MSE error for the Decission Tree Regressor is 0.022141440157720658 1.0

```
plt.figure(figsize=(60,15))  # set plot size (denoted in inches)
plot_tree(dec_tree,fontsize=20,max_depth=4)
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```



C: cross validation

```
flow = Pipeline([('decision_tree',DecisionTreeRegressor(random_state=0))])
mse = make_scorer(mean_squared_error,greater_is_better=False)
cv = GridSearchCV(estimator = flow,
                  scoring = mse,
                  param_grid={'decision_tree__max_depth':[*range(1,15)],
                         'decision_tree__min_samples_split':[*range(1,101)],
                         'decision_tree__min_samples_leaf':[*range(0,51)]})
```

```
cv.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.
10500 fits failed out of a total of 357000.
The score on these train-test partitions for these parameters will be set t
If these failures are not expected, you can try to debug them by setting er

Below are more details about the failures:
--------------------------------------------------------------------------------
7000 fits failed with the following error:
Traceback (most recent call last):
```

3.2 5 / 5

✓ **- 0 pts** **Correct**

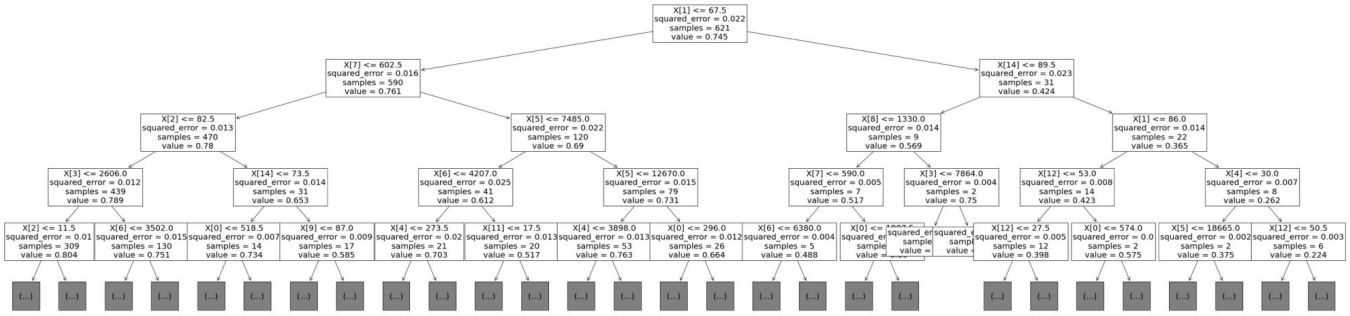
- 5 pts Not Done 3(b): Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

- 2 pts Didn't plot the tree properly, and interpret the results.

- 1 pts Test MSE not reported.

```
plt.figure(figsize=(60,15))  # set plot size (denoted in inches)
plot_tree(dec_tree,fontsize=20,max_depth=4)
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```



C: cross validation

```
flow = Pipeline([('decision_tree',DecisionTreeRegressor(random_state=0))])
mse = make_scorer(mean_squared_error,greater_is_better=False)
cv = GridSearchCV(estimator = flow,
                  scoring = mse,
                  param_grid={'decision_tree__max_depth':[*range(1,15)],
                         'decision_tree__min_samples_split':[*range(1,101)],
                         'decision_tree__min_samples_leaf':[*range(0,51)]})
```

```
cv.fit(X_train,y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.
10500 fits failed out of a total of 357000.
The score on these train-test partitions for these parameters will be set t
If these failures are not expected, you can try to debug them by setting er

Below are more details about the failures:
--------------------------------------------------------------------------------
7000 fits failed with the following error:
Traceback (most recent call last):
```

```
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_val
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/pipeline.py", line 3
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", l
    X_idx_sorted=X_idx_sorted,
  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", l
    % self.min_samples_leaf
ValueError: min_samples_leaf must be at least 1 or in (0, 0.5], got 0

--------------------------------------------------------------------------------
3500 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_val
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/pipeline.py", line 3
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", l
    X_idx_sorted=X_idx_sorted,
  File "/usr/local/lib/python3.7/dist-packages/sklearn/tree/_classes.py", l
    % self.min_samples_split
ValueError: min_samples_split must be an integer greater than 1 or a float

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:9
 -0.01689982]
  category=UserWarning,
GridSearchCV(estimator=Pipeline(steps=[('decision_tree',

DecisionTreeRegressor(random_state=0))]),
             param_grid={'decision_tree__max_depth': [1, 2, 3, 4, 5, 6, 7,
8, 9,
                                                      10, 11, 12, 13, 14],
                         'decision_tree__min_samples_leaf': [0, 1, 2, 3,
4, 5,
                                                             6, 7, 8, 9,
10, 11,
                                                             12, 13, 14,
15, 16,
                                                             17, 18, 19,
20, 21,
                                                             22, 23, 24,
25, 26,
                                                             27, 28, 29,
...],
                         'decision_tree__min_samples_split': [1, 2, 3, 4,
5, 6,
                                                              7, 8, 9, 10,
11,
                                                              12, 13, 14,
15
```

```
cv.best_estimator_.get_params
```

```
<bound method Pipeline.get_params of Pipeline(steps=[('decision_tree',
                 DecisionTreeRegressor(max_depth=3, min_samples_leaf=3,
                                       random_state=0))])>
```

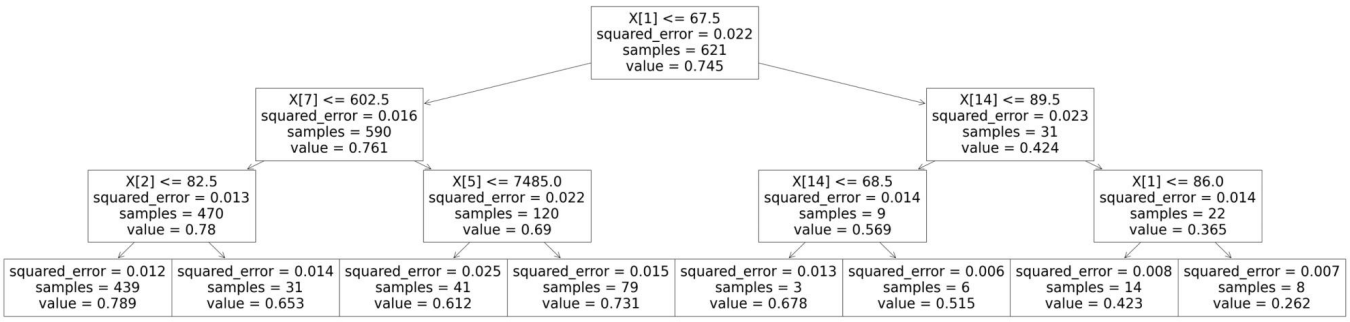
```
dec_tree_ = DecisionTreeRegressor(max_depth=3,min_samples_leaf=3,min_samples_spl
```

```
y_pred = dec_tree_.predict(X_test)
print('The MSE error for the Decission Tree Regressor post Cross Validation is {
mse_dec_tree_cv = mean_squared_error(y_true = y_test, y_pred=y_pred)
```

```
The MSE error for the Decission Tree Regressor post Cross Validation is 0.0
```

```
plt.figure(figsize=(60,15))  # set plot size (denoted in inches)
plot_tree(dec_tree_)
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```




```
dec_tree_.feature_importances_
```

```
array([0.        , 0.63587065, 0.09772537, 0.        , 0.        ,
       0.06902561, 0.        , 0.1390023 , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.05837607,
       0.        ])
```

The MSE error for the Decission Tree Regressor post Cross Validation is 0.01531490562
max_depth=3, min_samples_leaf=3, min_sample_leaf = 3

D: Bagging

```
bag_reg = BaggingRegressor(base_estimator = DecisionTreeRegressor()).fit(X_train
```

```
y_pred = bag_reg.predict(X_test)
print("MSE for the following problem is {}".
      format(mean_squared_error(y_true=y_test,y_pred=y_pred)))
mse_bag_reg = mean_squared_error(y_true=y_test,y_pred=y_pred)
```

```
    MSE for the following problem is 0.012177210816684237
```

```
print("mse decision tree {:.5f}, mse decision tree post
CV {:.5f} mse decision tree with bagging {:.5f}".
format(mse_dec_tree,mse_dec_tree_cv,mse_bag_reg))
```

```
    mse decision tree 0.02407, mse decision tree post CV 0.01531 mse decision t
```

```
bag_reg.base_estimator.fit(X_train,y_train).feature_importances_
```

```
    array([3.69794116e-02, 2.88017743e-01, 7.75976623e-02, 4.87401404e-02,
           3.82477558e-02, 1.21011640e-01, 1.01612654e-01, 8.45703466e-02,
           2.71281351e-02, 2.73890031e-02, 8.19571075e-03, 2.89401591e-02,
           3.22647416e-02, 1.95664859e-02, 5.95578160e-02, 1.80594586e-04])
```

3.3 10 / 10

✓ **- 0 pts** **Correct**

- 10 pts Not Done 3(c): Use cross-validation in order to determine the optimal depth of tree max_depth. Similarly, select the minimum number of samples for a split min_samples_split and for a leaf min_samples_leaf using cross-validation. Does selecting these parameters in the regression tree improve the test MSE? Plot the tree again and compare it to the plot from Step (b)

- 5 pts Cross-validation not done

- 2 pts Comment on Findings not done: Does selecting these parameters in the regression tree improve the test MSE? Plot the tree again and compare it to the plot from Step (b)

- 3 pts Tree plot not visible

The MSE error for the Decission Tree Regressor post Cross Validation is 0.01531490562
max_depth=3, min_samples_leaf=3, min_sample_leaf = 3

D: Bagging

```
bag_reg = BaggingRegressor(base_estimator = DecisionTreeRegressor()).fit(X_train
```

```
y_pred = bag_reg.predict(X_test)
print("MSE for the following problem is {}".
      format(mean_squared_error(y_true=y_test,y_pred=y_pred)))
mse_bag_reg = mean_squared_error(y_true=y_test,y_pred=y_pred)
```

```
    MSE for the following problem is 0.012177210816684237
```

```
print("mse decision tree {:.5f}, mse decision tree post
CV {:.5f} mse decision tree with bagging {:.5f}".
format(mse_dec_tree,mse_dec_tree_cv,mse_bag_reg))
```

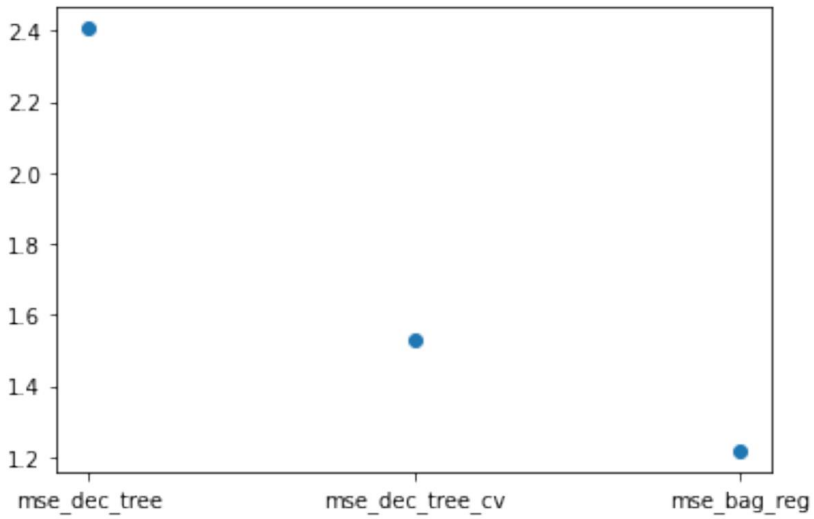
```
    mse decision tree 0.02407, mse decision tree post CV 0.01531 mse decision t
```

```
bag_reg.base_estimator.fit(X_train,y_train).feature_importances_
```

```
    array([3.69794116e-02, 2.88017743e-01, 7.75976623e-02, 4.87401404e-02,
           3.82477558e-02, 1.21011640e-01, 1.01612654e-01, 8.45703466e-02,
           2.71281351e-02, 2.73890031e-02, 8.19571075e-03, 2.89401591e-02,
           3.22647416e-02, 1.95664859e-02, 5.95578160e-02, 1.80594586e-04])
```

```
loss = {'mse_dec_tree':mse_dec_tree*10**2,
        'mse_dec_tree_cv':mse_dec_tree_cv*10**2,
        'mse_bag_reg':mse_bag_reg*10**2}
plt.scatter(x=loss.keys(),y=loss.values())
```

```
<matplotlib.collections.PathCollection at 0x7ff3768d9810>
```



Above I have compared all three loss MSE with Bagging being least 0.0121

E: Random Forest

```
estimators = [150,200,250,300,400,500,600,700,800,900]
features = [0.1,0.2,0.30,0.4,0.5,0.6,0.7,0.8,0.9,1]

random_reg = RandomForestRegressor(random_state=0).fit(X_train,y_train)


y_pred = random_reg.predict(X_test)
print("MSE through default Random forest parameters is {}".
    format(mean_squared_error(y_true=y_test,y_pred=y_pred)))
```

```
MSE through default Random forest parameters is 0.011270620593682 46
```

3.4 3 / 5

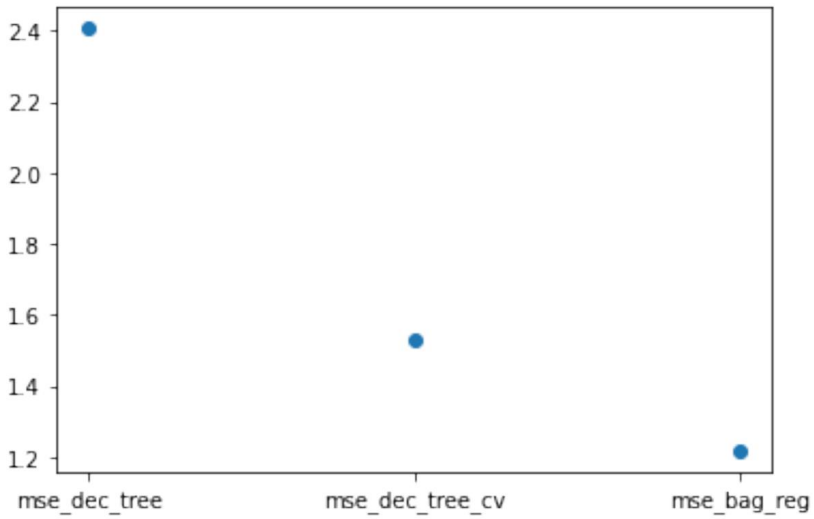
- 0 pts Correct

- 5 pts Not Done 3(d): Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the feature_importances_() attribute in DecisionTreeRegressor() to identify which variables are most important.

✓ **- 2 pts Not stated the feature_importances_() attribute in DecisionTreeRegressor() to identify which variables are most important.**

```
loss = {'mse_dec_tree':mse_dec_tree*10**2,
        'mse_dec_tree_cv':mse_dec_tree_cv*10**2,
        'mse_bag_reg':mse_bag_reg*10**2}
plt.scatter(x=loss.keys(),y=loss.values())
```

```
<matplotlib.collections.PathCollection at 0x7ff3768d9810>
```



Above I have compared all three loss MSE with Bagging being least 0.0121

E: Random Forest

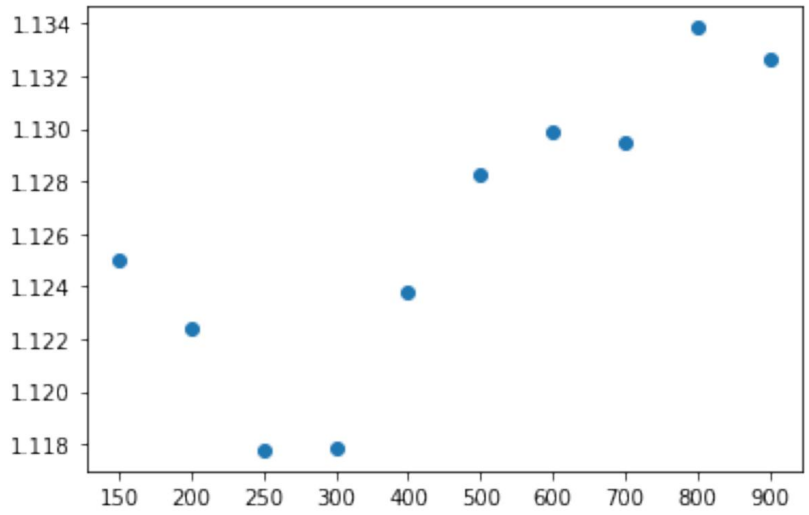
```
estimators = [150,200,250,300,400,500,600,700,800,900]
features = [0.1,0.2,0.30,0.4,0.5,0.6,0.7,0.8,0.9,1]

random_reg = RandomForestRegressor(random_state=0).fit(X_train,y_train)


y_pred = random_reg.predict(X_test)
print("MSE through default Random forest parameters is {}".
    format(mean_squared_error(y_true=y_test,y_pred=y_pred)))
```

```
MSE through default Random forest parameters is 0.011270620593682 46
```

```
mse = {}
for i in estimators:
  random_reg = RandomForestRegressor(n_estimators=i,max_features=1.0,
                                     random_state=0).fit(X_train,y_train)
  y_pred = random_reg.predict(X_test)
  print(random_reg.get_params )
  mse[str(i)] = mean_squared_error(y_true = y_test, y_pred=y_pred)*10**2
```

```
plt.scatter(x=mse.keys(),y = mse.values())
```

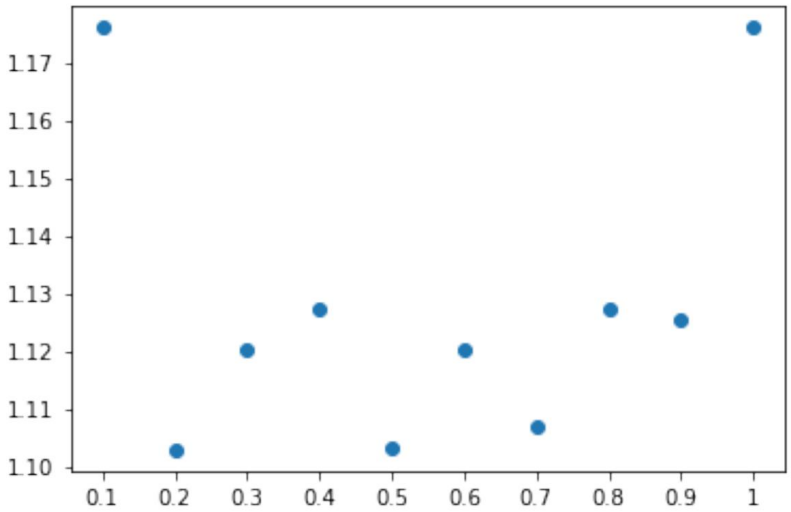
```
<matplotlib.collections.PathCollection at 0x7ff374691d50>
```




```
mse = {}
for i in features:
  random_reg = RandomForestRegressor(n_estimators=250,max_features=i,
                                     random_state=0).fit(X_train,y_train)
  y_pred = random_reg.predict(X_test)
  mse[str(i)] = mean_squared_error(y_true = y_test, y_pred=y_pred)*10**2
```

```
plt.scatter(x=mse.keys(),y = mse.values())
```

```
<matplotlib.collections.PathCollection at 0x7ff370187e50>
```




```
random_reg = RandomForestRegressor(n_estimators=250,max_features=0.2,random_stat
y_pred = random_reg.predict(X_test)
print('The MSE with number of trees as 250 and number of features as 0.2 gives
the lowest mse {}'.format(mean_squared_error(y_true = y_test, y_pred=y_pred)))
mse_random_forest = mean_squared_error(y_true = y_test, y_pred=y_pred)*10**2
```

```
The MSE with number of trees as 250 and number of features as 50 gives the
```

```
random_reg.feature_importances_
```

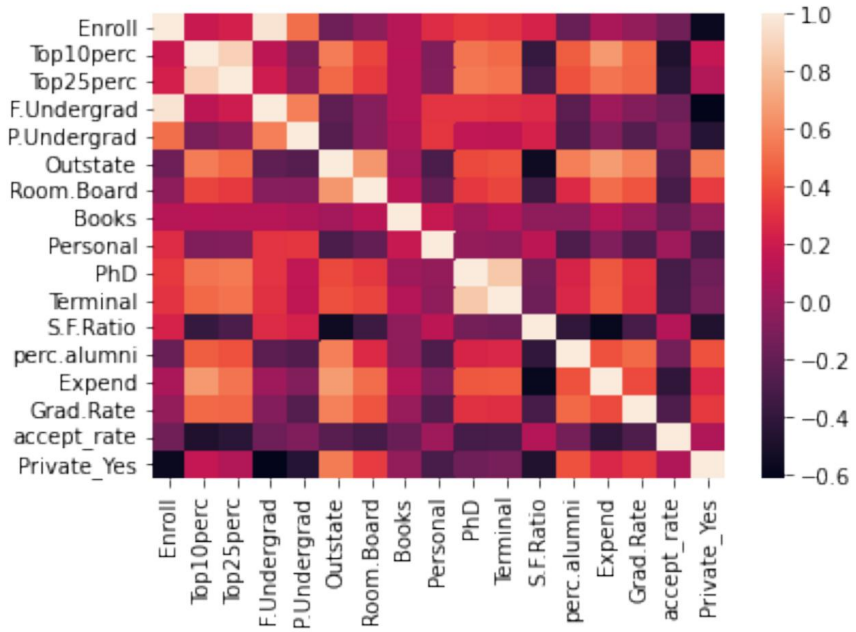
```
array([0.04623837, 0.12041184, 0.11018919, 0.05232979, 0.04358619,
       0.1027753 , 0.07324469, 0.05557029, 0.03883525, 0.04738192,
       0.04184073, 0.05001879, 0.0403998 , 0.07153076, 0.09550515,
       0.01014193])
```

```
X_train.columns
```

```
Index(['Enroll', 'Top10perc', 'Top25perc', 'F.Undergrad', 'P.Undergrad',
       'Outstate', 'Room.Board', 'Books', 'Personal', 'PhD', 'Terminal',
       'S.F.Ratio', 'perc.alumni', 'Expend', 'Grad.Rate', 'Private_Yes'],
      dtype='object')
```

```
sns.heatmap(data.corr(method = 'pearson'))
#makes sense
```

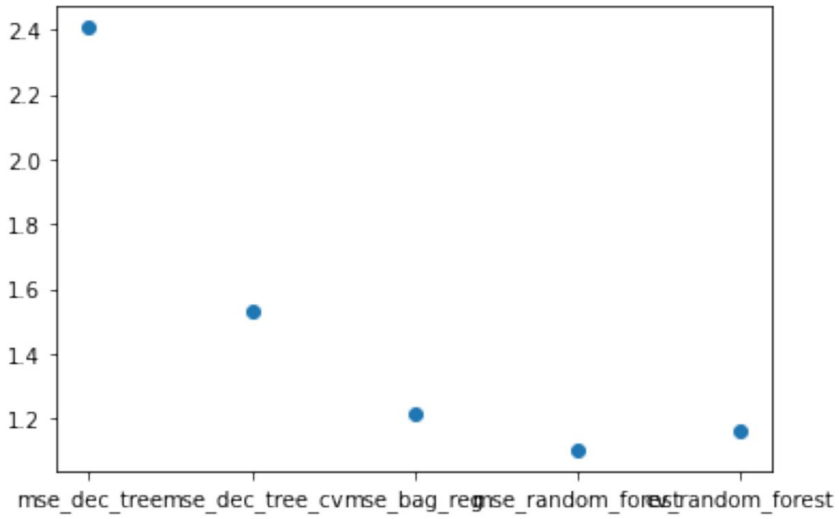
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff373c48190>
```




```
#GridCV to find and confirm our guess


mse = make_scorer(mean_squared_error, greater_the_better = False)
param_grid = {'random_forest__n_estimators':[150,200,250,300,400,500,600,700,800
              'random_forest__max_features':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9

flow = Pipeline([('random_forest',RandomForestRegressor(random_state=0))])

cv_random_forest = GridSearchCV(estimator=  flow,
                                scoring = mse,
                                param_grid=param_grid)


cv_random_forest.fit(X_train,y_train)
```

```
loss['mse_random_forest'] = mse_random_forest
loss['cv_random_forest'] = mse_cv_random_forest
plt.scatter(x=loss.keys(),y=loss.values())
```

```
<matplotlib.collections.PathCollection at 0x7ff36feb7750>
```



As we can see with number of trees 250 and 0.5 features we get the least mse of 0.011 among all the above models

3.5 5 / 5

✓ **- 0 pts** **Correct**

- 5 pts Not Done 3(e) : Use random forests to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important. Describe the effect of number of trees m as well as the number of variables considered at each split, on the error rate obtained.

- 2 pts Not or incorrectly described the effect of number of trees m as well as the number of variables considered at each split, on the error rate obtained.

```
import pandas as pd
import numpy as np
from sklearn.utils import resample
from sklearn.tree import DecisionTreeClassifier
import sklearn.metrics as sm
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
import random
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt
import seaborn as sns


data = pd.read_csv('penguins_size.csv')
data = data.dropna()


data.isnull().sum()
```

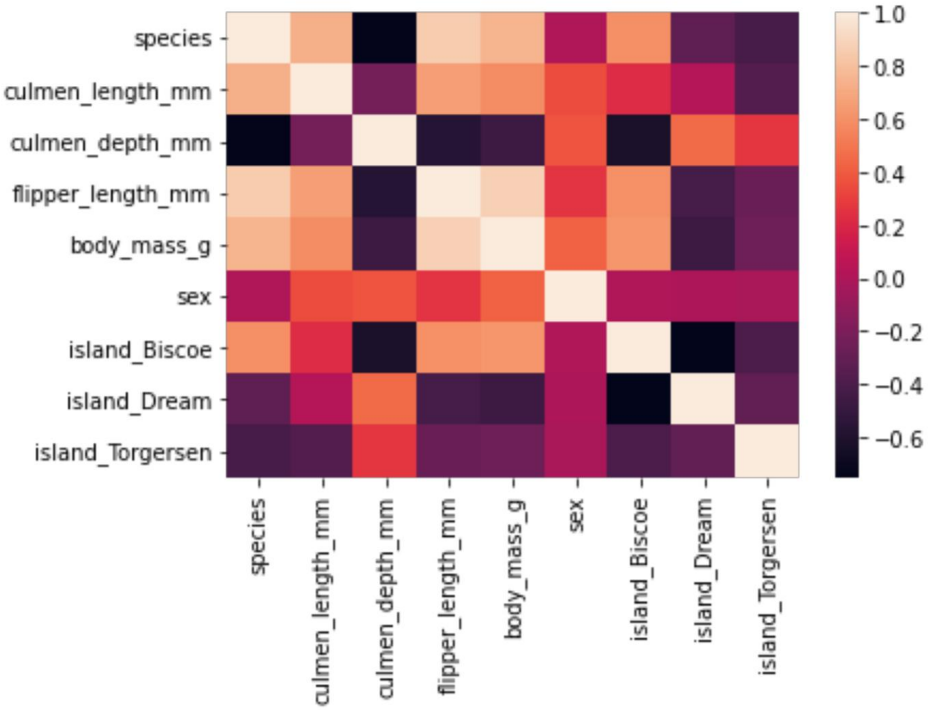
```
species              0
island               0
culmen_length_mm     0
culmen_depth_mm      0
flipper_length_mm    0
body_mass_g          0
sex                  0
dtype: int64
```

```
data.head()
```

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	b
0	Adelie	Torgersen	39.1	18.7	181.0	
1	Adelie	Torgersen	39.5	17.4	186.0	
2	Adelie	Torgersen	40.3	18.0	195.0	
4	Adelie	Torgersen	36.7	19.3	193.0	
5	Adelie	Torgersen	39.3	20.6	190.0	

```
sns.heatmap(data.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f96888780d0>
```




```
data.island.unique()
```

```
array(['Torgersen', 'Biscoe', 'Dream'], dtype=object)
```

```
data.species.unique()
```

```
array(['Adelie', 'Chinstrap', 'Gentoo'], dtype=object)
```

```
data.sex.unique()
```

```
array(['MALE', 'FEMALE', '.'], dtype=object)
```

```
data = data[data.sex != '.']
data.species = data.species.astype('category').cat.codes
data.sex = data.sex.astype('category').cat.codes
data = pd.get_dummies(data,columns=['island'])
```

```
data.head()
```

	species	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_
0	0	39.1	18.7	181.0	3750.
1	0	39.5	17.4	186.0	3800.
2	0	40.3	18.0	195.0	3250.
4	0	36.7	19.3	193.0	3450.
5	0	39.3	20.6	190.0	3650.

A

```
from sklearn.model_selection import train_test_split

data_train,data_test = train_test_split(data, test_size=0.2,shuffle=True)

boot_sample = resample(data_train,replace=True,n_samples=100)

y_boot_sample = boot_sample.pop('species')
X_boot_sample =boot_sample
X_boot_sample.head()
#y_train = data_train.pop('species')
#X_train = data_train
```

	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
147	36.6	18.4	184.0	3475.0	0
201	49.8	17.3	198.0	3675.0	0
104	37.9	18.6	193.0	2925.0	0
144	37.3	16.8	192.0	3000.0	0
321	55.9	17.0	228.0	5600.0	1

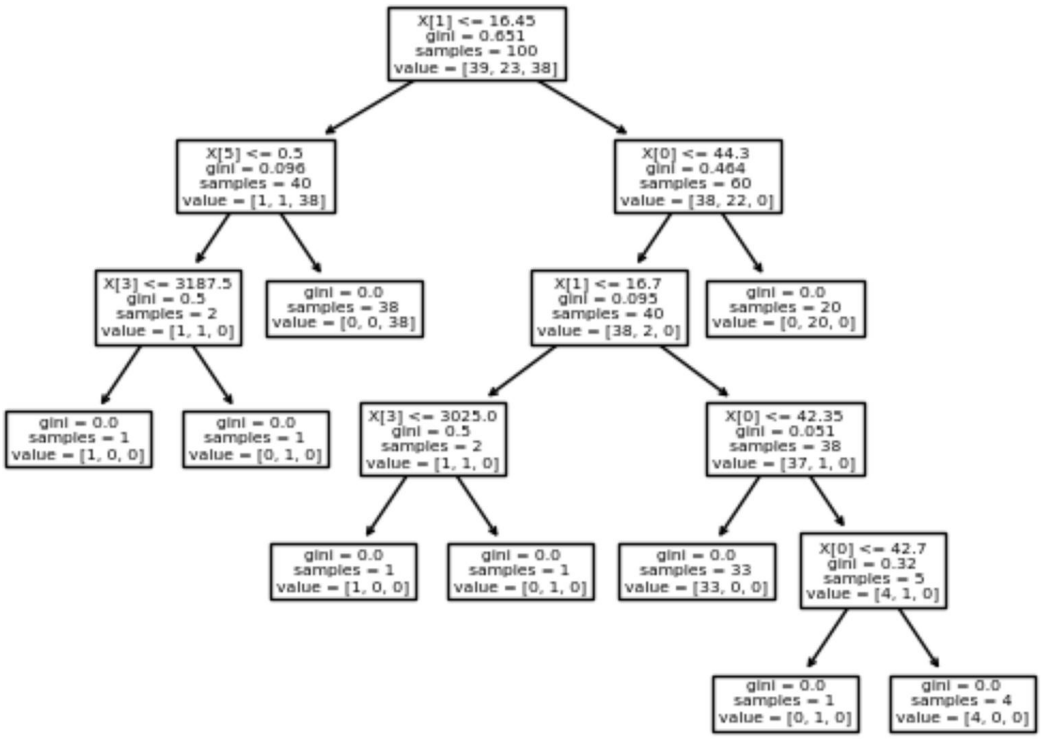
```
data_train.head()
```

	species	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mas
46	0	41.1	19.0	182.0	342
178	1	50.5	18.4	200.0	340
108	0	38.1	17.0	181.0	317
149	0	37.8	18.1	193.0	375
298	2	45.2	13.8	215.0	475

```
p_scores = []
p_models = []
for i in range(1,len(X_boot_sample.columns)+1):
    sample_i_p = X_boot_sample.sample(n = i,axis=1)
    model_i = DecisionTreeClassifier()
    scores_i = cross_val_score(model_i,sample_i_p,y_boot_sample,scoring='neg_mea
    p_scores.append(-1 * scores_i.mean())
    p_models.append(list(sample_i_p.columns))


results = pd.DataFrame()
results['Models'] = p_models
results['MSE'] = p_scores
results = results.sort_values('MSE')


results['p'] = results.Models.apply(len)
results
```

	Models	MSE	p
1	[culmen_length_mm, island_Biscoe]	0.08	2
5	[culmen_length_mm, island_Torgersen, sex, isla...	0.13	6
4	[culmen_length_mm, sex, island_Biscoe, culmen_...	0.16	5
6	[culmen_length_mm, island_Torgersen, sex, flip...	0.18	7
7	[island_Biscoe, island_Torgersen, sex, flipper...	0.20	8
2	[flipper_length_mm, sex, island_Biscoe]	0.21	3
3	[island_Biscoe, culmen_depth_mm, island_Torger...	0.21	4
0	[island_Torgersen]	1.27	1

```
from sklearn.tree import DecisionTreeRegressor,plot_tree
tree_model = DecisionTreeClassifier(max_depth=5,max_features=5)
tree_model = tree_model.fit(X_boot_sample,y_boot_sample)
plot_tree(tree_model)
```



P=2 gives the lowest cross validation error

```
from IPython.display import Image, display
from sklearn.tree import export_graphviz
from sklearn import tree
import graphviz
```

B

4.1 5 / 5

✓ **- 0 pts** Correct

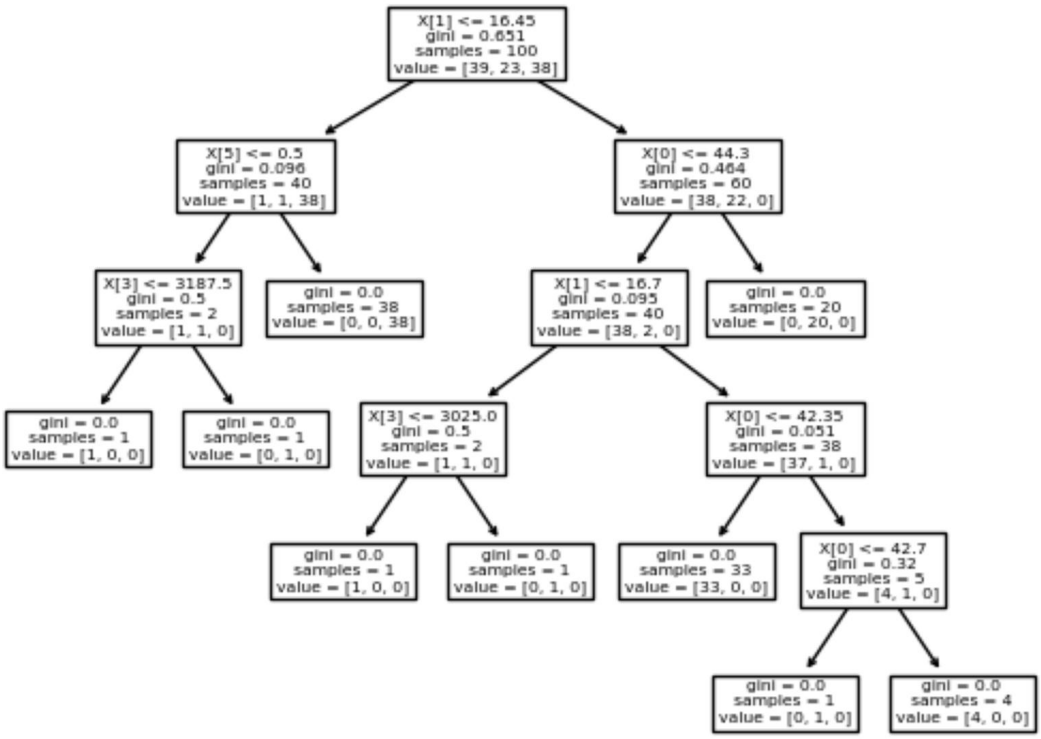
- 5 pts Not Done 4(a): Begin by creating a bootstrap sample of size 100 and then select a subset of p columns. Vary the value of p and report the p that results in the lowest cross-validation error. Now train a decision tree classifier on the bootstrap sample by setting the depth to 6.

- 2 pts Incorrect Bootstrap sampling of size 100

- 2 pts Incorrect selection of a subset of p columns. Vary the value of p and report the p that results in the lowest cross-validation error.

- 1 pts Incorrect training of decision tree classifier on the bootstrap sample by setting the depth to 6.

```
from sklearn.tree import DecisionTreeRegressor,plot_tree
tree_model = DecisionTreeClassifier(max_depth=5,max_features=5)
tree_model = tree_model.fit(X_boot_sample,y_boot_sample)
plot_tree(tree_model)
```



P=2 gives the lowest cross validation error

```
from IPython.display import Image, display
from sklearn.tree import export_graphviz
from sklearn import tree
import graphviz
```

B

```
from collections import Counter
def my_rf(n,train):
    n_models = []
    train_preditions = []
    test_predictions =[]
    train_prob = []
    test_prob =[]

    for i in range(n):
        sample_i = resample(train,replace=True,n_samples=100)
        y_sample_i = sample_i.pop('species')
        X_sample_i = sample_i
        model = DecisionTreeClassifier(max_features=random.randint(1,len(X_sampl
        model_i = model.fit(X_sample_i,y_sample_i)
        test_i_prediction = model_i.predict(data_test.loc[:,data_test.columns !=
        train_i_prediction = model_i.predict(data_train.loc[:,data_train.columns

        test_i_probprediction = model_i.predict_proba(data_test.loc[:,data_test.
        train_i_probprediction = model_i.predict_proba(data_train.loc[:,data_tra
        n_models.append(model_i)

        train_preditions.append(train_i_prediction)
        test_predictions.append(test_i_prediction)

        train_prob.append(train_i_probprediction)
        test_prob.append(test_i_probprediction)

    train_preditions = np.array(train_preditions)
    test_predictions = np.array(test_predictions)

    train_prob = np.array(train_prob)
    test_prob = np.array(test_prob)
    train_majority = np.array([np.bincount(m).argmax() for m in train_preditions
    test_majority = np.array([np.bincount(i).argmax() for i in test_predictions.

    train_avg_prob = np.mean(train_prob,axis=0)
    test_avg_prob = np.mean(test_prob,axis=0)
    return train_majority,test_majority,train_avg_prob,test_avg_prob
```

```
T = [1,50,100,150,200,300,400]
p_names = []
p_accuracy = []

for t in T:
    y_pred_t = my_rf(t,data_train)[0]
    accuracy = sm.accuracy_score(y_pred_t,data_train['species'])
    p_accuracy.append(accuracy)
    p_names.append('T_'+str(t))
```

C

```
results = pd.DataFrame()
p_names = []
test_p_accuracy = []
train_p_accuracy = []
test_p_f1 = []
train_p_f1 = []
train_p_auc = []
test_p_auc = []
for t in T:
    train_pred_t,test_pred_t,train_proba_t,test_proba_t = my_rf(t,data_train)

    train_accuracy = sm.accuracy_score(train_pred_t,data_train['species'])
    test_accuracy = sm.accuracy_score(test_pred_t,data_test['species'])

    train_f1 = sm.f1_score(train_pred_t,data_train['species'],average = 'micro')
    test_f1 = sm.f1_score(test_pred_t,data_test['species'],average = 'micro')

    train_auc = sm.roc_auc_score(data_train['species'],train_proba_t,multi_class
    test_auc = sm.roc_auc_score(data_test['species'],test_proba_t,multi_class='o

    test_p_accuracy.append(test_accuracy)
    train_p_accuracy.append(train_accuracy)

    test_p_f1.append(test_f1)
    train_p_f1.append(train_f1)

    test_p_auc.append(test_auc)
    train_p_auc.append(train_auc)

    p_names.append('T_'+str(t))
results['T'] = p_names
results['Train_Accuracy'] = train_p_accuracy
results['test_accuracy'] = test_p_accuracy
```

4.2 5 / 5

✓ **- 0 pts** **Correct**

- 5 pts Not done 4(b): Repeat the above step to generate T ∈ {1,50,100,150,200,300,400} trees and evaluate your training set. Combine the predictions from all trees and assign the final class based on a majority vote of the predictions of every tree. In the case of ties, assign a class randomly among the ties.

- 2 pts Incorrect T ∈ {1,50,100,150,200,300,400} trees generation and evaluate on the training set.

- 2 pts Incorrect aggregation of predictions from all trees and assign the final class based on a majority vote of the predictions of every tree.

- 1 pts In the case of ties, incorrectly assigned a class among the ties.

- 1 pts The evaluation metric is not as expected.

```
T = [1,50,100,150,200,300,400]
p_names = []
p_accuracy = []

for t in T:
    y_pred_t = my_rf(t,data_train)[0]
    accuracy = sm.accuracy_score(y_pred_t,data_train['species'])
    p_accuracy.append(accuracy)
    p_names.append('T_'+str(t))
```

C

```
results = pd.DataFrame()
p_names = []
test_p_accuracy = []
train_p_accuracy = []
test_p_f1 = []
train_p_f1 = []
train_p_auc = []
test_p_auc = []
for t in T:
    train_pred_t,test_pred_t,train_proba_t,test_proba_t = my_rf(t,data_train)

    train_accuracy = sm.accuracy_score(train_pred_t,data_train['species'])
    test_accuracy = sm.accuracy_score(test_pred_t,data_test['species'])

    train_f1 = sm.f1_score(train_pred_t,data_train['species'],average = 'micro')
    test_f1 = sm.f1_score(test_pred_t,data_test['species'],average = 'micro')

    train_auc = sm.roc_auc_score(data_train['species'],train_proba_t,multi_class
    test_auc = sm.roc_auc_score(data_test['species'],test_proba_t,multi_class='o

    test_p_accuracy.append(test_accuracy)
    train_p_accuracy.append(train_accuracy)

    test_p_f1.append(test_f1)
    train_p_f1.append(train_f1)

    test_p_auc.append(test_auc)
    train_p_auc.append(train_auc)

    p_names.append('T_'+str(t))
results['T'] = p_names
results['Train_Accuracy'] = train_p_accuracy
results['test_accuracy'] = test_p_accuracy
```

```
results['Train_f1'] = train_p_f1
results['test_f1'] = test_p_f1
results['Train_auc'] = train_p_auc
results['test_auc'] = test_p_auc

results
```

	T	Train_Accuracy	test_accuracy	Train_f1	test_f1	Train_auc	test_a
0	T_1	0.969925	0.910448	0.969925	0.910448	0.972816	0.9285
1	T_50	0.996241	0.985075	0.996241	0.985075	0.999666	1.0000
2	T_100	0.992481	0.985075	0.992481	0.985075	0.999874	1.0000
3	T_150	0.996241	0.985075	0.996241	0.985075	0.999927	1.0000
4	T_200	0.992481	0.970149	0.992481	0.970149	0.999962	1.0000
5	T_300	0.996241	0.985075	0.996241	0.985075	0.999962	1.0000
6	T_400	0.992481	0.970149	0.992481	0.970149	0.999991	1.0000

Above is the error obtained with different trees

D

```
results = pd.DataFrame()
p_names = []
test_p_accuracy = []
train_p_accuracy = []
test_p_f1 = []
train_p_f1 = []
train_p_auc = []
test_p_auc = []
T = [10,50,100]
for t in T:
    model = RandomForestClassifier(n_estimators=t).fit(data_train.loc[:,data_tra
                                                       data_train['species'])
    train_pred_t = model.predict(data_train.loc[:,data_train.columns !='species'
    test_pred_t = model.predict(data_test.loc[:,data_test.columns !='species'])
    train_proba_t = model.predict_proba(data_train.loc[:,data_train.columns !='s
    test_proba_t = model.predict_proba(data_test.loc[:,data_test.columns !='spec

    train_accuracy = sm.accuracy_score(train_pred_t,data_train['species'])
    test_accuracy = sm.accuracy_score(test_pred_t,data_test['species'])

    train_f1 = sm.f1_score(train_pred_t,data_train['species'],average = 'micro')
```

4.3 5 / 5

✓ **- 0 pts** **Correct**

- 5 pts Not done 4(c): Report the training and test error, F1 score, and AUC by varying T in the range {1,50,100,150,200,300,400}.

- 3 pts Not trained the model with the given mentioned trees.

- 2 pts Incorrect/incomplete evaluation metric

```
results['Train_f1'] = train_p_f1
results['test_f1'] = test_p_f1
results['Train_auc'] = train_p_auc
results['test_auc'] = test_p_auc

results
```

	T	Train_Accuracy	test_accuracy	Train_f1	test_f1	Train_auc	test_a
0	T_1	0.969925	0.910448	0.969925	0.910448	0.972816	0.9285
1	T_50	0.996241	0.985075	0.996241	0.985075	0.999666	1.0000
2	T_100	0.992481	0.985075	0.992481	0.985075	0.999874	1.0000
3	T_150	0.996241	0.985075	0.996241	0.985075	0.999927	1.0000
4	T_200	0.992481	0.970149	0.992481	0.970149	0.999962	1.0000
5	T_300	0.996241	0.985075	0.996241	0.985075	0.999962	1.0000
6	T_400	0.992481	0.970149	0.992481	0.970149	0.999991	1.0000

Above is the error obtained with different trees

D

```
results = pd.DataFrame()
p_names = []
test_p_accuracy = []
train_p_accuracy = []
test_p_f1 = []
train_p_f1 = []
train_p_auc = []
test_p_auc = []
T = [10,50,100]
for t in T:
    model = RandomForestClassifier(n_estimators=t).fit(data_train.loc[:,data_tra
                                                       data_train['species'])
    train_pred_t = model.predict(data_train.loc[:,data_train.columns !='species'
    test_pred_t = model.predict(data_test.loc[:,data_test.columns !='species'])
    train_proba_t = model.predict_proba(data_train.loc[:,data_train.columns !='s
    test_proba_t = model.predict_proba(data_test.loc[:,data_test.columns !='spec

    train_accuracy = sm.accuracy_score(train_pred_t,data_train['species'])
    test_accuracy = sm.accuracy_score(test_pred_t,data_test['species'])

    train_f1 = sm.f1_score(train_pred_t,data_train['species'],average = 'micro')
```

```
    test_f1 = sm.f1_score(test_pred_t,data_test['species'],average = 'micro')

    train_auc = sm.roc_auc_score(data_train['species'],
                                 train_proba_t,multi_class='ovr')
    test_auc = sm.roc_auc_score(data_test['species'],
                                test_proba_t,multi_class='ovr')

    test_p_accuracy.append(test_accuracy)
    train_p_accuracy.append(train_accuracy)

    test_p_f1.append(test_f1)
    train_p_f1.append(train_f1)

    test_p_auc.append(test_auc)
    train_p_auc.append(train_auc)

    p_names.append('T_'+str(t))
results['T'] = p_names
results['Train_Accuracy'] = train_p_accuracy
results['test_accuracy'] = test_p_accuracy
results['Train_f1'] = train_p_f1
results['test_f1'] = test_p_f1
results['Train_auc'] = train_p_auc
results['test_auc'] = test_p_auc

results
```

	T	Train_Accuracy	test_accuracy	Train_f1	test_f1	Train_auc	test_a
0	T_10	0.996241	1.0	0.996241	1.0	1.0	
1	T_50	1.000000	1.0	1.000000	1.0	1.0	
2	T_100	1.000000	1.0	1.000000	1.0	1.0	

```
perm_importance = permutation_importance(model,
      data_train.loc[:,data_train.columns !='species'], data_train['species'])
sorted_idx = perm_importance.importances_mean.argsort()
```

```
sorted_idx
```

```
    array([7, 4, 5, 1, 3, 2, 6, 0])
```

Cumen-length, Flipper-lenth has the most influence on the model

4.4 5 / 5

✓ **- 0 pts** **Correct**

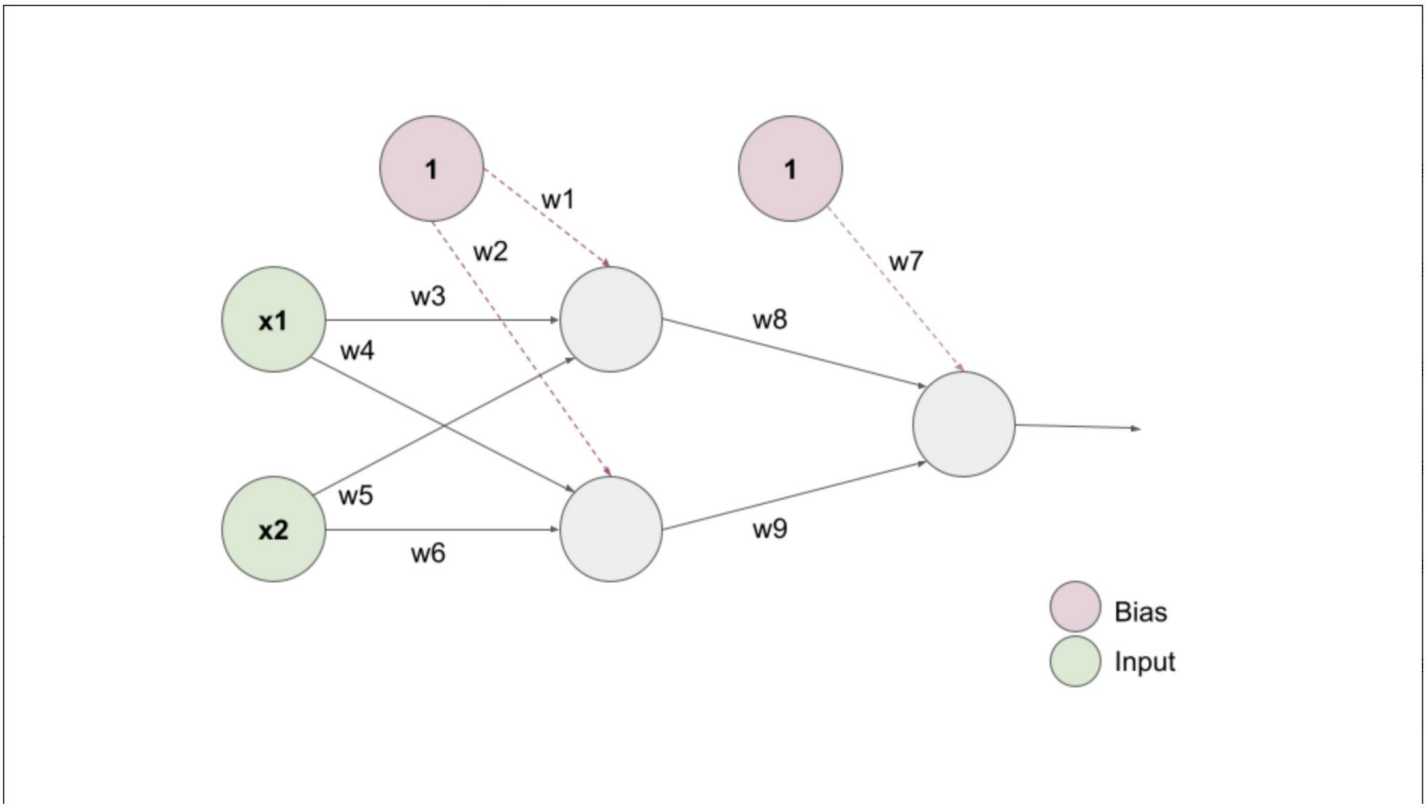
- 5 pts Not Done 4(d): Use an existing package to train a Random Forest algorithm with 10, 50, and 100 decision trees. Report similar metrics on both the training and testing sets. Report the top features having the most influence on the model.

- 2 pts Not trained a Random Forest algorithm with 10/50/100 decision trees.

- 2 pts Incorrectly/incompletely reported metrics on both the training and testing sets.

- 1 pts Incorrectly reported the top 5 features having the most influence on the model.

Part A



Let Z1 and Z2 be the two hidden nodes, we know that the hidden layer uses a linear activation function f(z') = c.z

for the first Neuron Z1 (before activation function)

Z1,1 = X1*W3* + *X2*W5 + W2 --------- eq. 1

W3,W5 being weight associated with the input neurons and W2 being the weight of the Bias.

Similarly,

Z2,1 = X1*W4* + *X2*W6 + W1 ---------- eq. 2

Now, these neurons will go through the activation function,

Z1,2 = C*Z1,1* ------- *eq.3 Z2,2 = C*Z2,1 ------- eq.4

Moving on with the forward propogation the output will output (o/p) will be as follows,

o/p' = Z1,2*W8* + *Z2,2*W9 +W7

The output is then fed into the sigmoid function the output for the case P(y=1|x,2)

o/p = o/p'/1+e^-(o/p')

Therefore from equation 1,2,3 and 4 we get,

$$O/P = \frac{1}{1 + e^{-[C*W8*(X1*W3+X2*W5+W2)+C*W9*(X1*W4+X2*W6+W1)+W7)]}}$$

The classification boundary will be for (y=1|x,w). we will have to maximize the below equation to get the desired output

C.W8(X1.W3 + X2.W5 + W2) + C.W9(X1.W4 + X2.W6 + W1)+ W7 -> ∞

PART B

5.1 **2 / 5**

- **0 pts** Correct

✓ - **1 pts** **incorrect classification boundary**

✓ - **2 pts** **incorrect output representation**

- **1 pts** in correct formula/calculation.

- **5 pts** no submission

1. For eq1, must be + W1 and for eq 2, + W2
2. o/p' =0
3. incorrect sigmoid function

Let Z1 and Z2 be the two hidden nodes, we know that the hidden layer uses a linear activation function f(z') = c.z

for the first Neuron Z1 (before activation function)

Z1,1 = X1*W3* + *X2*W5 + W2 --------- eq. 1

W3,W5 being weight associated with the input neurons and W2 being the weight of the Bias.

Similarly,

Z2,1 = X1*W4* + *X2*W6 + W1 ---------- eq. 2

Now, these neurons will go through the activation function,

Z1,2 = C*Z1,1* ------- *eq.3 Z2,2 = C*Z2,1 ------- eq.4

Moving on with the forward propogation the output will output (o/p) will be as follows,

o/p' = Z1,2*W8* + *Z2,2*W9 +W7

The output is then fed into the sigmoid function the output for the case P(y=1|x,2)

o/p = o/p'/1+e^-(o/p')

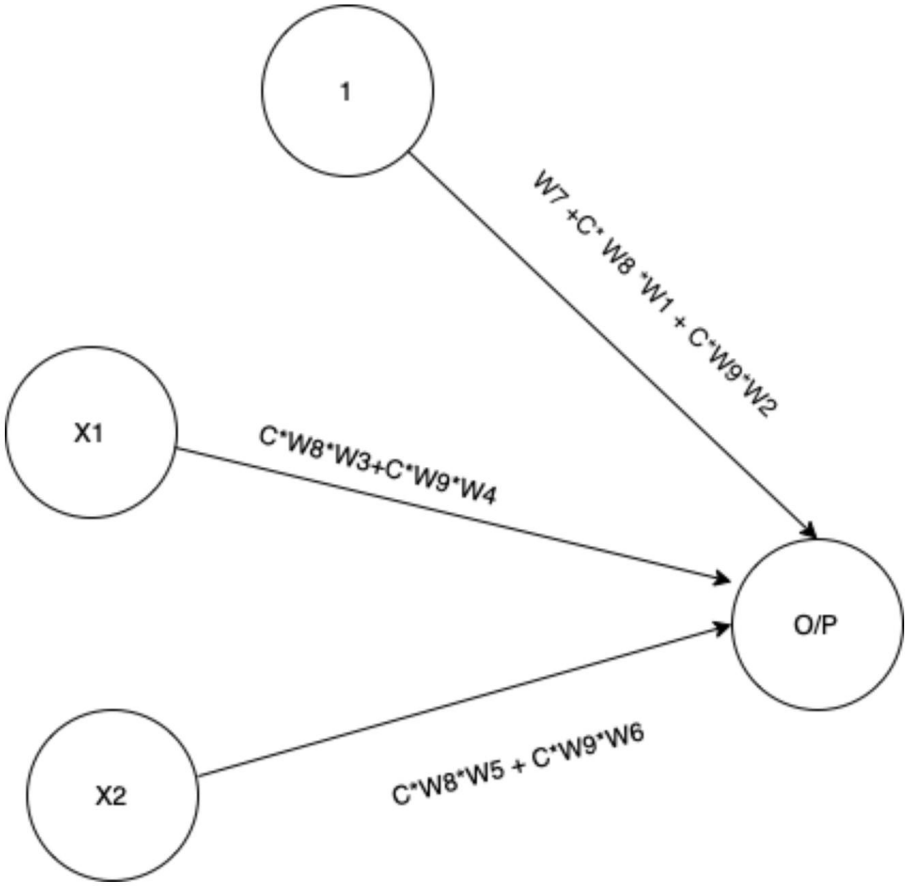
Therefore from equation 1,2,3 and 4 we get,

$$O/P = \frac{1}{1 + e^{-[C*W8*(X1*W3+X2*W5+W2)+C*W9*(X1*W4+X2*W6+W1)+W7)]}}$$

The classification boundary will be for (y=1|x,w). we will have to maximize the below equation to get the desired output

C.W8(X1.W3 + X2.W5 + W2) + C.W9(X1.W4 + X2.W6 + W1)+ W7 -> ∞

PART B



We can represent any multilayer neural network by a neural network without any hidden layer provided the original hidden layer had a **Linear activation Function** since the output will be linear combination of input features weights and biases.

5.2 5 / 5

✓ **- 0 pts** Correct

- 2 pts missing neural network diagram

- 2 pts incorrect explaination

- 1 pts vague /mistake in explanation

- 1 pts mistake in weights

- 2 pts no explaination mentioned

- 1 pts mistake in diagram

- 5 pts no submission