Exam 2

Honer Code Pledge

By submitting your answers to this exam, you acknowledge adherence to the Honor Code Pledge:

On my honor as a University of Colorado Boulder student I have neither given nor received unauthorized assistance

Background

We have learned:

- Frequent Patterns
- Classification
- Regression
- Clustering

We learned how to analyze data for patterns. We have played with many tutorials to learn the tools as well.

In this exam, you will be given a dataset, and you are going to apply the methods and tools to it. To receive full credits, **besides showing and explaining the result the methods, you should also explain the rational of why and how you tuned the models (why and how you chose hyperparameters)**..

Best results will be published like a data science competition

Read and Understand the data (5 points)

In this part, you should import the data, use stats and visualization tools, to get a basic understanding of it.

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

```
df = pd.read_csv("student_record_10k.csv")
```

```
df.head()
```

	Unnamed: 0 int64	id int64	major object	gender object	c01 float64	c02
0	0	0	Computer Science	M	75.7575757575757 5	82
1	1	1	Electric Engineering	M	81.8181818181818 1	90
2	2	2	Computer Science	M	81.2121212121212 2	83
3	3	3	Computer Science	F	82.424242424242 42	85.
4	4	4	Computer Science	M	86.666666666666 67	88

```
df["elective"].head()
```

```
0    ['Databases' 'Statistical_Inference'\n 'High_P...
1    ['Databases' 'Data_Mining' 'Big_Data' 'Nature_...
2    ['Text_Marketing_Analysis' 'Databases' 'Stats_...
3    ['Effective_Communication' 'Data_Structures_an...
4    ['R_for_Data_Science' 'Databases' 'Machine_Lea...
Name: elective, dtype: object
```

```
df.info()
```

```
---  ------               --------------  -----
 0   Unnamed: 0           10000 non-null  int64
 1   id                   10000 non-null  int64
 2   major                10000 non-null  object
 3   gender               10000 non-null  object
 4   c01                  10000 non-null  float64
 5   c02                  10000 non-null  float64
 6   c03                  10000 non-null  float64
 7   c04                  10000 non-null  float64
 8   c05                  10000 non-null  float64
 9   c06                  10000 non-null  float64
 10  c07                  10000 non-null  float64
```

```
 15  campus             10000 non-null  int64
 16  internship         10000 non-null  int64
 17  AtRisk_academic    10000 non-null  int64
 18  AtRisk_campus      10000 non-null  int64
 19  AtRisk_internship  10000 non-null  int64
 20  AtRisk             10000 non-null  int64
 21  graduate_program   10000 non-null  float64
 22  goverment          10000 non-null  float64
 23  industry           10000 non-null  float64
 24  placement          10000 non-null  float64
 25  annual             10000 non-null  float64
 26  elective           10000 non-null  object
dtypes: float64(16), int64(8), object(3)
```

Elective, Major, and Gender are the only objects in this dataset.

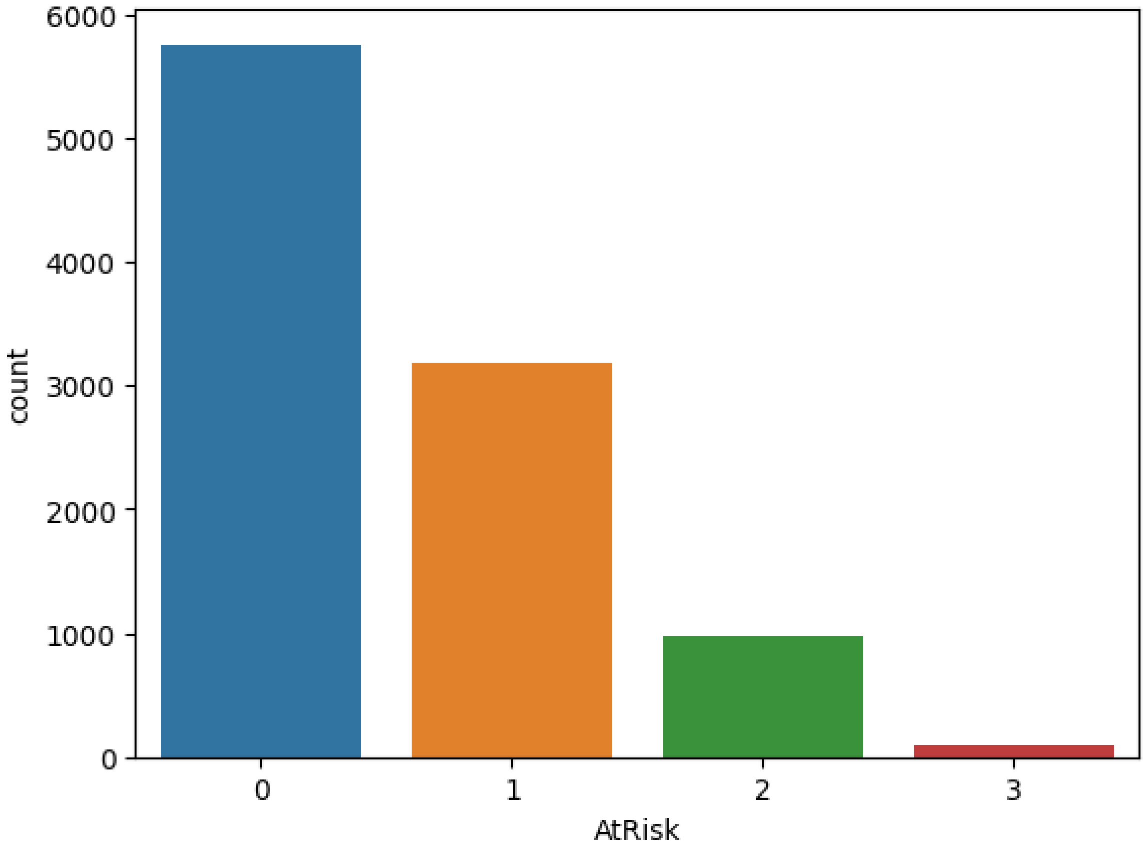
```
df.describe()
```


	Unnamed: 0 float...	id float64	c01 float64	c02 float64	c03 float64	c04
			5	9	8	
std	2886.8956799071675	2886.8956799071675	6.611917718096645	5.481640360463034	7.824067170518313	11.4
min	0.0	0.0	60.0	69.76744186046511	53.48837209302326	
25%	2499.75	2499.75	76.36363636363636	81.3953488372093	69.18604651162791	
50%	4999.5	4999.5	80.60606060606061	84.88372093023256	75.0	
75%	7499.25	7499.25	85.45454545454545	88.95348837209302	80.23255813953489	
max	9999.0	9999.0	100.0	100.0	100.0	

The mean for AtRisk is .53 and the std is .7. This indicates that the majority of students are not at a high level of risk.

It's also worth noting that the mean for academic performance is 78.9%

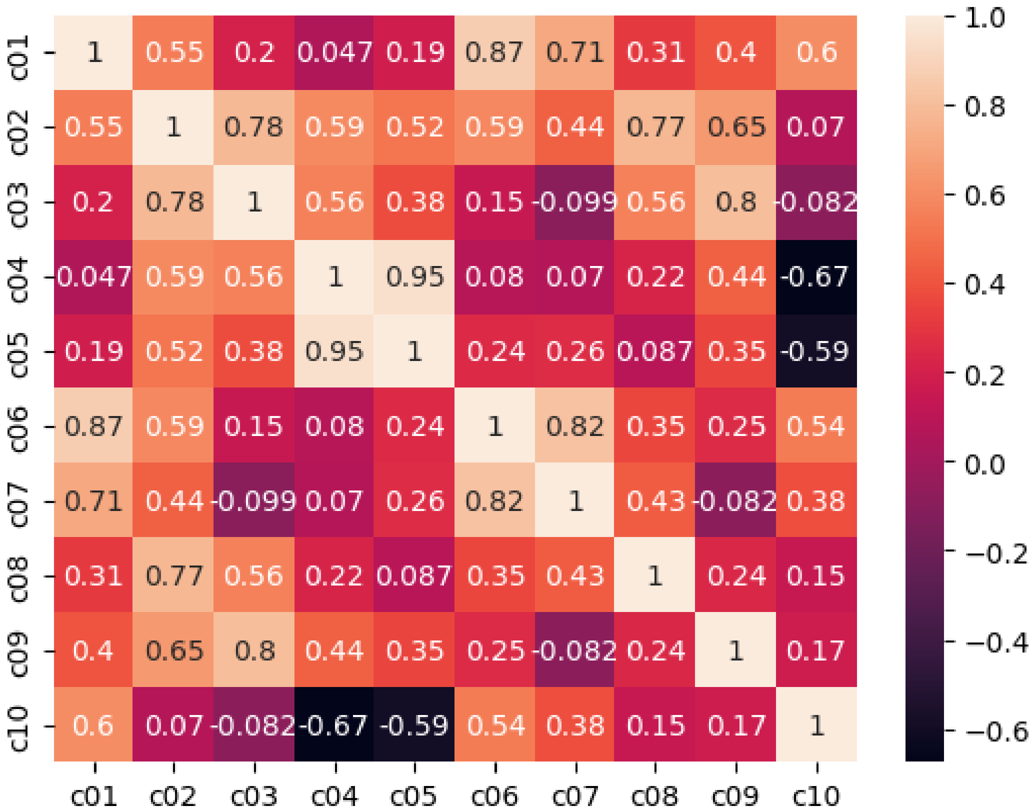
```
sns.countplot(data=df, x='AtRisk')
plt.show()
```



This is confirmed by the histogram below where we see the majority of students fall within low levels of risk

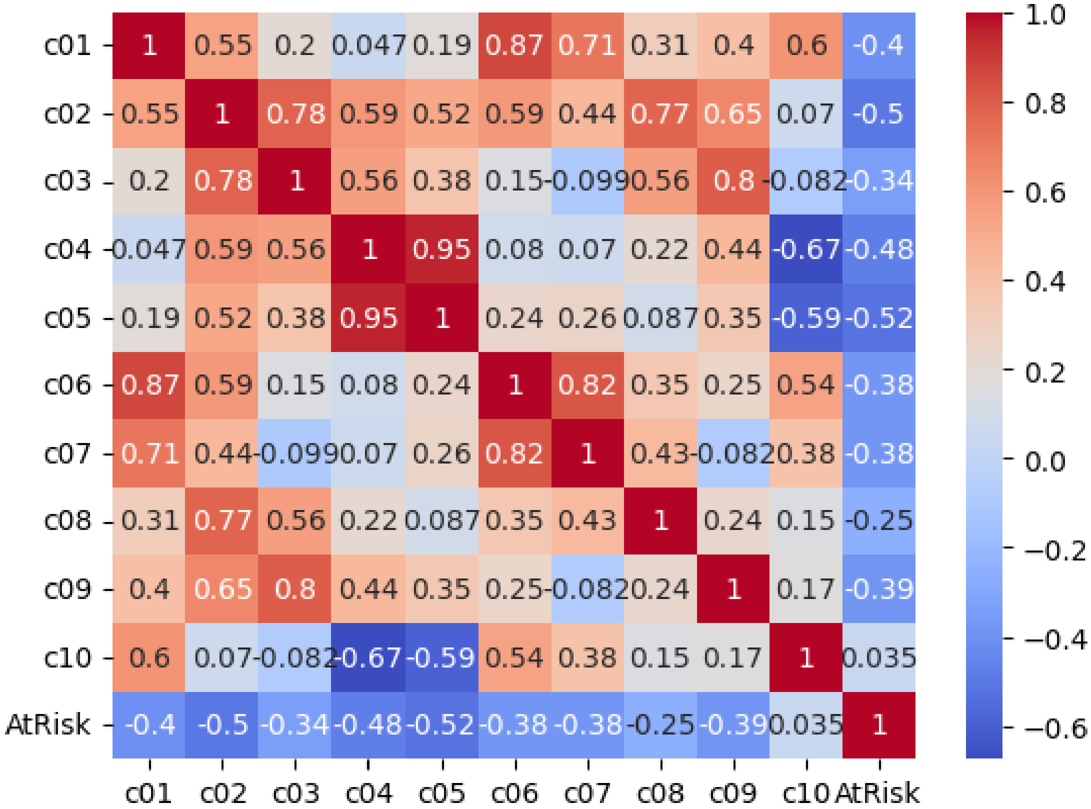
Now I want to look at the correlation of class scores

```
class_corr = df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10']].corr
sns.heatmap(class_corr, annot=True)
plt.show()
```

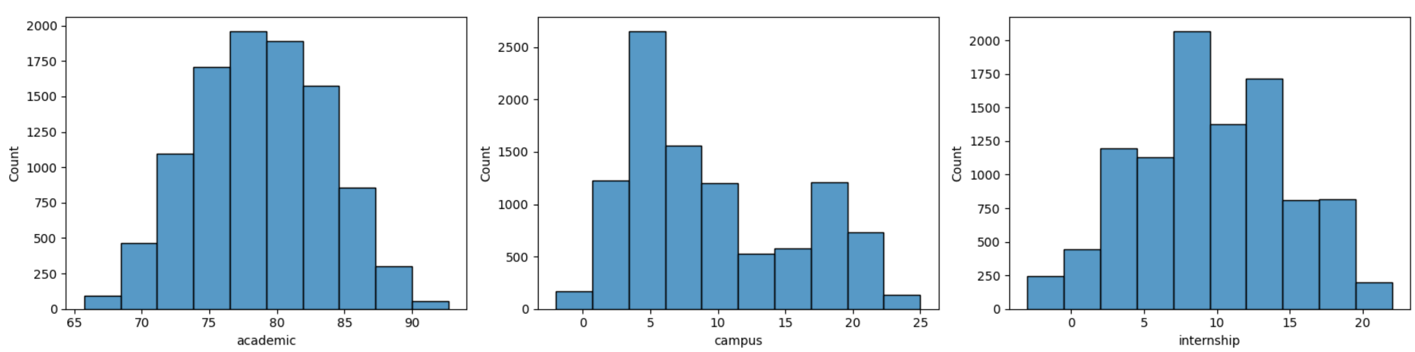


This heatmap looks for classes scores that are related to AtRisk

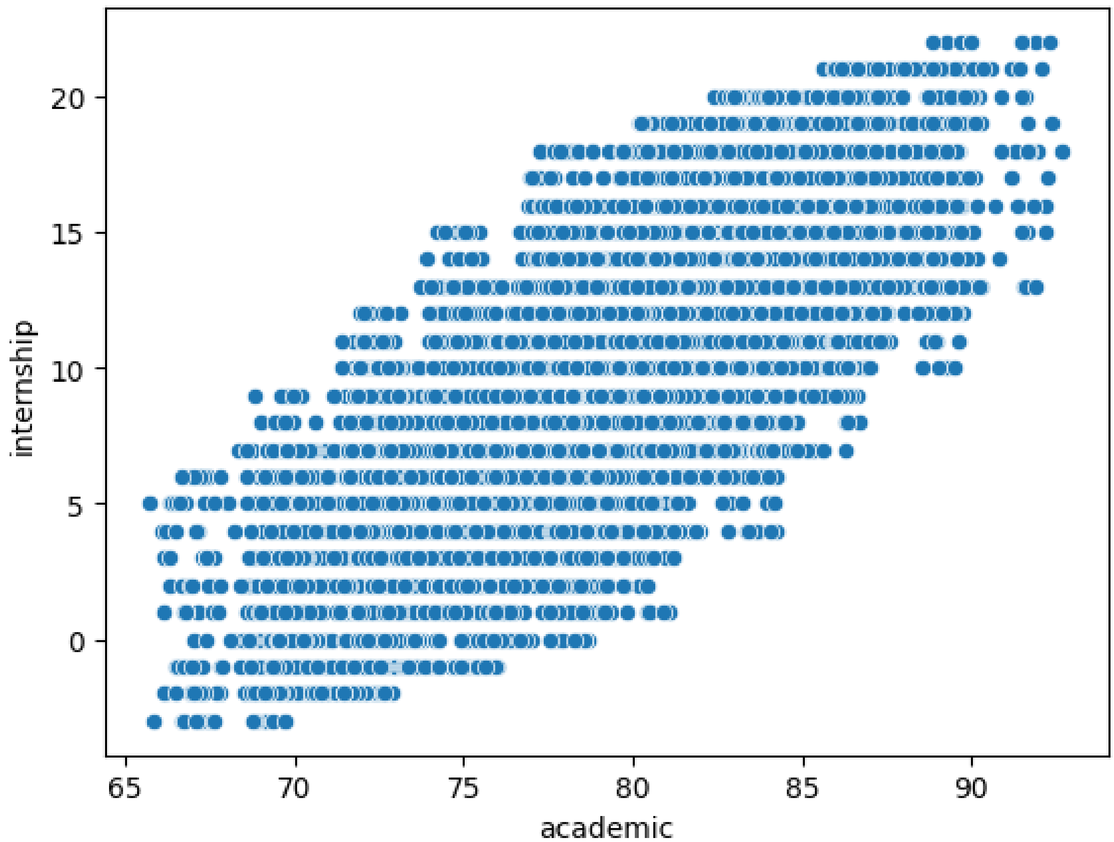
```
corr_matrix = df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10', 'AtR
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True)
plt.show()
```




```
fig, axs = plt.subplots(1, 3, figsize=(16, 4))
sns.histplot(data=df, x='academic', bins=10, ax=axs[0])
sns.histplot(data=df, x='campus', bins=10, ax=axs[1])
sns.histplot(data=df, x='internship', bins=10, ax=axs[2])
axs[0].set_xlabel('academic')
axs[1].set_xlabel('campus')
axs[2].set_xlabel('internship')
plt.tight_layout()
plt.show()
```




```
sns.scatterplot(data=df, x='academic', y='internship')
plt.show()
```



Mean scores grouped by major

```
df.groupby('major')[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10']].
```

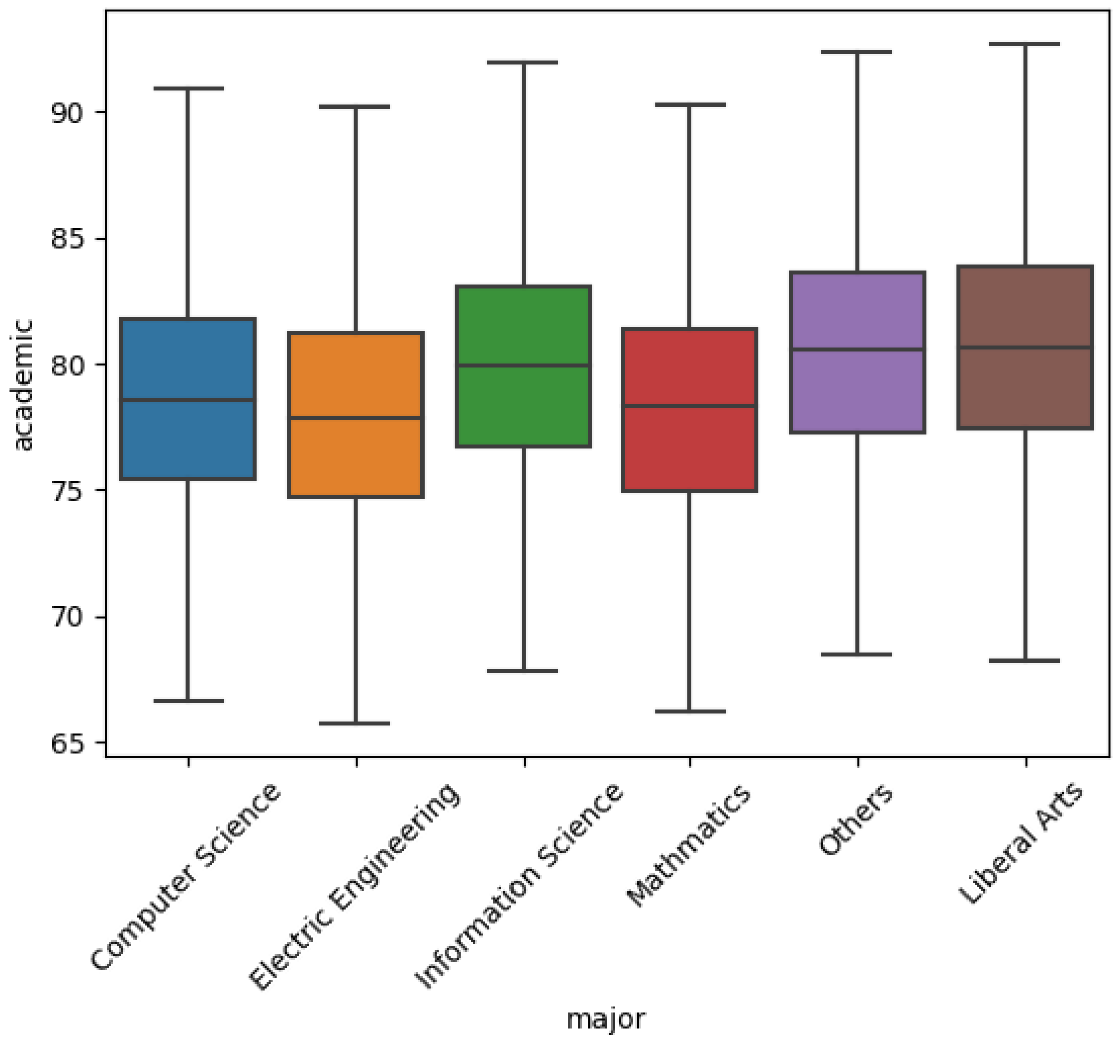
	c01 mean float64	c02 mean float64	c03 mean float64	c04 mean float64	c05 mean float64	c06
Computer...	83.54149837403435	83.60687659953281	69.1451143715525	64.64183185235817	68.82907136564786	85.
Electric Engin...	78.13017540735446	86.58690750062641	79.58453864200635	64.69368266405485	61.64640032247016 4	76.
Information...	76.13621362136209	87.01013124568279	84.12848261570348	88.18712871287129	85.06380240912173	72
Liberal Arts	80.11333333333333	86.21860465116272	75.10174418604643	86.777	88.88989169675091	80
Mathmatics	83.6013330233279	82.6075060964731	72.83381906857768	63.732736572890026	67.01210448078159	82
Others	78.25679475164026	87.85880865717837	76.19657374512325	85.3120899718838	85.17521036408978	79

Dispersion statistics for each class

```
print("Range:\n", df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10']]
print("Variance:\n", df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10
print("Standard Deviation:\n", df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', '
print("25th quantile:\n", df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09',
print("50th quantile:\n", df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09',
print("75th quantile:\n", df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09',
print("IQR:\n", df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10']].c
```

```
c05    69.675090
c06    80.645161
c07    82.894737
c08    81.818182
c09    78.971963
c10    82.627119
Name: 0.5, dtype: float64
75th quantile:
 c01    85.454545
c02    88.953488
c03    80.232558
c04    82.000000
c05    81.588448
c06    86.635945
c07    87.500000
c08    86.363636
c09    84.112150
c10    87.711864
Name: 0.75, dtype: float64
IQR:
 c01     9.090909
c02     7.558140
c03    11.046512
c04    19.500000
c05    17.328520
c06    11.981567
c07     9.210526
c08     9.090909
c09     9.345794
c10    17.796610
dtype: float64
```

```
ax = sns.boxplot(data=df, x='major', y='academic')
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)
plt.show()
```



Classification (30 points)

AtRisk (10 points)

Use columns 'c01', 'c02', ..., 'c10','academic', 'campus', and 'internship', and use *Decision Tree* to classify students who need attention. The column 'AtRisk' measures the level of attention needed, 0 stands for no need, 3 stands for a lot attention needed. As the hypterparameter training, you should set proper argument for the Decision Tree algorithm.

Let's see how the model performs using default parameters

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

X_train, X_test, y_train, y_test = train_test_split(
    df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10', 'academic', 'c
```

```python
    df['AtRisk'], test_size=0.2, random_state=2)

# Train a Decision Tree Classifier with default hyperparameters
clf = DecisionTreeClassifier(random_state=2)
clf.fit(X_train, y_train)

# Predict train w/ accuracy score
y_train_pred = clf.predict(X_train)
train_acc_score = accuracy_score(y_train, y_train_pred)
print("Training accuracy score:", train_acc_score)

# training set
print("Confusion matrix (training set):")
print(confusion_matrix(y_train, y_train_pred))
print("Classification report (training set):")
print(classification_report(y_train, y_train_pred))

# Predict test w/ accuracy score
y_test_pred = clf.predict(X_test)
test_acc_score = accuracy_score(y_test, y_test_pred)
print("Testing accuracy score:", test_acc_score)

# test set
print("Confusion matrix (test set):")
print(confusion_matrix(y_test, y_test_pred))
print("Classification report (test set):")
print(classification_report(y_test, y_test_pred))
```

```
Classification report (training set):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4604
           1       1.00      1.00      1.00      2540
           2       1.00      1.00      1.00       790
           3       1.00      1.00      1.00        66

    accuracy                           1.00      8000
   macro avg       1.00      1.00      1.00      8000
weighted avg       1.00      1.00      1.00      8000

Testing accuracy score: 0.9995
Confusion matrix (test set):
[[1151    0    0    0]
```

```
           0       1.00      1.00      1.00      1151
           1       1.00      1.00      1.00       637
           2       1.00      0.99      1.00       192
           3       1.00      1.00      1.00        20

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

I was extremely surprised by the results of this classification and had worries about overfitting. Since many metrics like accuracy, f1, precision, and recall are very similar throughout the testing and training this is an indication that the model is not overfitting. However, it is important to note that the complexity of this model is very high so it is possible that it is just learning the noise. On the other hand, I learned from my correlation matrix that there are many subjects that are highly correlated to AtRisk.

Next, I am going to use a Grid search and Kfold validation to evaluate my model on multiple splits of data and find the best hyperparameters for this model.

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold


X_train, X_test, y_train, y_test = train_test_split(
    df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10', 'academic', 'c
    df['AtRisk'], test_size=0.2, random_state=2)

# Define the parameter grid to search over
param_opts = {'max_depth': [1, 2, 3],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 2, 4],
              'max_features': [None, 'sqrt', 'log2'],
              'criterion': ['gini', 'entropy']}

# Define the cross-validation strategy
cross_val = KFold(n_splits=5, shuffle=True, random_state=2)


# Define the grid search object
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=2), param_opts, cv=cross_val,

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and accuracy score
print("Best hyperparameters:", grid_search.best_params_)
print("Train accuracy score:", grid_search.best_score_)
```

```python
# Evaluate the performance on the test set
y_pred = grid_search.predict(X_test)
acc_score = accuracy_score(y_test, y_pred)
print("Test accuracy score:", acc_score)
```

```
Best hyperparameters: {'criterion': 'gini', 'max_depth': 3, 'max_features': None, 'min_samples_leaf': 1, 'r
Train accuracy score: 0.99975
Test accuracy score: 0.9995
```

The code above iteratively checks all of the combinations of possible hyperparameters within the parameter grid and chooses the best combination based on the accuracy score. Kfold will split the data into 5 different datasets to ensure that it will generalize well and avoid issues with overfitting.

```python
clf = DecisionTreeClassifier(criterion="gini", max_depth=3, max_features=None, min_samples_l
                            min_samples_split=2, random_state=2)
clf.fit(X_train, y_train)

# # Predict train w/ accuracy score
y_train_pred = clf.predict(X_train)
train_acc_score = accuracy_score(y_train, y_train_pred)
print("Training accuracy score:", train_acc_score)

# training set
print("Confusion matrix (training set):")
print(confusion_matrix(y_train, y_train_pred))
print("Classification report (training set):")
print(classification_report(y_train, y_train_pred))

# Predict test w/ accuracy score
y_test_pred = clf.predict(X_test)
test_acc_score = accuracy_score(y_test, y_test_pred)
print("Testing accuracy score:", test_acc_score)

# test set
print("Confusion matrix (test set):")
print(confusion_matrix(y_test, y_test_pred))
print("Classification report (test set):")
print(classification_report(y_test, y_test_pred))
```

```
Classification report (training set):
              precision    recall  f1-score   support
```

```
   macro avg       1.00      1.00      1.00      8000
weighted avg       1.00      1.00      1.00      8000

Testing accuracy score: 0.9995
Confusion matrix (test set):
[[1151    0    0    0]
 [   0  637    0    0]
 [   0    1  191    0]
 [   0    0    0   20]]
Classification report (test set):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1151
           1       1.00      1.00      1.00       637
           2       1.00      0.99      1.00       192
           3       1.00      1.00      1.00        20

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

graduate program (10 points)

Use columns 'c01', 'c02', ..., 'c10','academic', 'campus', and 'internship', and use *Logistic Regression* to classify students who will continue in a graduate program. The column 'graduate_program' indicates the likihood of the student who will continue in a graduate program. 0 means impossible, and 1 means very possible.

```
# create new column changing graduate program from continuous to binary
df['graduate_program_binary'] = df['graduate_program'].apply(lambda x: 1 if x>=0.5 else 0)
```

```
from sklearn.linear_model import LogisticRegression

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10', 'academic', 'c
    df['graduate_program_binary'], test_size=0.2, random_state=2)

# Train a logistic regression classifier
clf = LogisticRegression(random_state=2)
clf.fit(X_train, y_train)

# # Predict train w/ accuracy score
y_train_pred = clf.predict(X_train)
```

```python
train_acc_score = accuracy_score(y_train, y_train_pred)
print("Training accuracy score:", train_acc_score)

# # Predict test w/ accuracy score
y_test_pred = clf.predict(X_test)
test_acc_score = accuracy_score(y_test, y_test_pred)
print("Testing accuracy score:", test_acc_score)

#  training set
print("Confusion matrix (training set):")
print(confusion_matrix(y_train, y_train_pred))
print("Classification report (training set):")
print(classification_report(y_train, y_train_pred))

# test set
print("Confusion matrix (test set):")
print(confusion_matrix(y_test, y_test_pred))
print("Classification report (test set):")
print(classification_report(y_test, y_test_pred))
```

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Training accuracy score: 0.88525
Testing accuracy score: 0.8845
Confusion matrix (training set):
[[4425  438]
 [ 480 2657]]
Classification report (training set):
              precision    recall  f1-score   support

           0       0.90      0.91      0.91      4863
           1       0.86      0.85      0.85      3137

    accuracy                           0.89      8000
   macro avg       0.88      0.88      0.88      8000
weighted avg       0.89      0.89      0.89      8000

Confusion matrix (test set):
[[1080  109]
 [ 122  689]]
Classification report (test set):
              precision    recall  f1-score   support
```

I tried using a GridSearch to find the best hyperparameters but this was too time-consuming so I switched to the random forest with a max combination of 5 so it wouldn't take too long.

```
from sklearn.model_selection import RandomizedSearchCV

param_dist = {
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'max_iter': [50, 100, 200, 500]
}

random_search = RandomizedSearchCV(
    estimator=LogisticRegression(random_state=2),
    param_distributions=param_dist,
    n_iter=5,  # Number of random combinations to try
    cv=5,  # Number of cross-validation folds
    scoring='accuracy',
    n_jobs=-1,  # Use all available CPU cores
    verbose=1,  # Show progress
    random_state=2
)


random_search.fit(X_train, y_train)

print("Best parameters found:", random_search.best_params_)
print("Best accuracy score found:", random_search.best_score_)


best_clf = random_search.best_estimator_

# Training set performance
y_train_pred = best_clf.predict(X_train)
train_acc_score = accuracy_score(y_train, y_train_pred)
print("Training accuracy score with best parameters:", train_acc_score)

# Test set performance
y_test_pred = best_clf.predict(X_test)
test_acc_score = accuracy_score(y_test, y_test_pred)
print("Testing accuracy score with best parameters:", test_acc_score)

# training set
print("Confusion matrix (training set):")
print(confusion_matrix(y_train, y_train_pred))
print("Classification report (training set):")
print(classification_report(y_train, y_train_pred))

# test set
```

```
print("Confusion matrix (test set):")
print(classification_report(y_test, y_test_pred))
```

```
  warnings.warn(some_fits_failed_message, FitFailedWarning)
/shared-libs/python3.9/py/lib/python3.9/site-packages/sklearn/model_selection/_search.py:953: UserWarning:
  warnings.warn(
Best parameters found: {'solver': 'liblinear', 'penalty': 'l2', 'max_iter': 200, 'C': 1000}
Best accuracy score found: 0.9953749999999999
Training accuracy score with best parameters: 0.997375
Testing accuracy score with best parameters: 0.9975
Confusion matrix (training set):
[[4852   11]
 [  10 3127]]
Classification report (training set):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      4863
           1       1.00      1.00      1.00      3137

    accuracy                           1.00      8000
   macro avg       1.00      1.00      1.00      8000
weighted avg       1.00      1.00      1.00      8000

Confusion matrix (test set):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1189
           1       1.00      1.00      1.00       811

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

```
from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split(
    df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10', 'academic', 'c
    df['graduate_program_binary'], test_size=0.2, random_state=2)

# Train a logistic regression classifier
clf = LogisticRegression(solver= 'liblinear',
                         penalty='l2',
                         max_iter= 200,
                         C= 1000,
                         random_state=2)
clf.fit(X_train, y_train)
```

```python
# Predict train w/ accuracy score
y_train_pred = clf.predict(X_train)
train_acc_score = accuracy_score(y_train, y_train_pred)
print("Training accuracy score:", train_acc_score)

# Predict test w/ accuracy score
y_test_pred = clf.predict(X_test)
test_acc_score = accuracy_score(y_test, y_test_pred)
print("Testing accuracy score:", test_acc_score)

# training set
print("Confusion matrix (training set):")
print(confusion_matrix(y_train, y_train_pred))
print("Classification report (training set):")
print(classification_report(y_train, y_train_pred))

# test set
print("Confusion matrix (test set):")
print(confusion_matrix(y_test, y_test_pred))
print("Classification report (test set):")
print(classification_report(y_test, y_test_pred))
```

```
Training accuracy score: 0.997375
Testing accuracy score: 0.9975
Confusion matrix (training set):
[[4852   11]
 [  10 3127]]
Classification report (training set):
              precision    recall  f1-score   support


           0       1.00      1.00      1.00      4863
           1       1.00      1.00      1.00      3137


    accuracy                           1.00      8000
   macro avg       1.00      1.00      1.00      8000
weighted avg       1.00      1.00      1.00      8000


Confusion matrix (test set):
[[1185    4]
 [   1  810]]
Classification report (test set):
              precision    recall  f1-score   support


           0       1.00      1.00      1.00      1189
           1       1.00      1.00      1.00       811


    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
```

```
weighted avg       1.00      1.00      1.00      2000
```

Best parameters found: {'solver': 'liblinear', 'penalty': 'l2', 'max_iter': 200, 'C': 1000}. This makes sense for a couple of reasons, first lib linear is recommended when you have a very high dimension dataset which we currently do have. with this lib linear solver, we are limited in the penalties that are compatible with that solver method so the default l2 works great. The max iterations before convergence is 200, which is 100 more than the default 100. I received a convergence warning that It did not fully converge but since my testing and training results were similar and high I decided it was ok. A high C value indicates that the regularization is weak in my model, but this may also contribute to possible overfitting since coefficients can take up larger values

placement (10 points)

Use columns 'c01', 'c02', ..., 'c10','academic', 'campus', and 'internship', and use *A METHOD OF YOUR CHOICE* to classify students who will have a placement. The column 'placement' measures the likelihood of students will get placements. 0 stands for no chance, and 3 means the student has high probability to get multiple placements. You should convert the value to binary: [0, 0.5) as False, and [0.5, 3] as True.

```
from sklearn.ensemble import RandomForestClassifier

# binary: [0, 0.5) as False, and [0.5, 3] as True
df['placement_binary'] = df['placement'].apply(lambda x: x >= 0.5)

X_train, X_test, y_train, y_test = train_test_split(
    df[['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10', 'academic', 'c
    df['placement_binary'], test_size=0.2, random_state=2)

# Random Forest classifier
clf = RandomForestClassifier(random_state=2)
clf.fit(X_train, y_train)

# Predict train w/ accuracy score
y_train_pred = clf.predict(X_train)
train_acc_score = accuracy_score(y_train, y_train_pred)
print("Training accuracy score:", train_acc_score)

# Predict test w/ accuracy score
y_test_pred = clf.predict(X_test)
test_acc_score = accuracy_score(y_test, y_test_pred)
print("Testing accuracy score:", test_acc_score)

# training set
print("Confusion matrix (training set):")
print(confusion_matrix(y_train, y_train_pred))
print("Classification report (training set):")
print(classification_report(y_train, y_train_pred))
```

```python
# test set
print("Confusion matrix (test set):")
print(confusion_matrix(y_test, y_test_pred))
print("Classification report (test set):")
print(classification_report(y_test, y_test_pred))
```

```
Training accuracy score: 1.0
Testing accuracy score: 0.9855
Confusion matrix (training set):
[[ 204    0]
 [   0 7796]]
Classification report (training set):
              precision    recall  f1-score   support

       False       1.00      1.00      1.00       204
        True       1.00      1.00      1.00      7796

    accuracy                           1.00      8000
   macro avg       1.00      1.00      1.00      8000
weighted avg       1.00      1.00      1.00      8000

Confusion matrix (test set):
[[  33   28]
 [   1 1938]]
Classification report (test set):
              precision    recall  f1-score   support

       False       0.97      0.54      0.69        61
        True       0.99      1.00      0.99      1939

    accuracy                           0.99      2000
   macro avg       0.98      0.77      0.84      2000
weighted avg       0.99      0.99      0.98      2000
```

I'll use random Forest again to see which hyperparameter combinations will work best

```python
# Define the parameter distribution for RandomizedSearchCV
param_dist = {
    'n_estimators': [10, 50, 100, 200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
```

```python
# Perform randomizedsearchCV to find the best hyperparameters
random_search = RandomizedSearchCV(
    estimator=RandomForestClassifier(random_state=2),
    param_distributions=param_dist,
    n_iter=10,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1,
    random_state=2
)

random_search.fit(X_train, y_train)

print("Best parameters found:", random_search.best_params_)
print("Best accuracy score found:", random_search.best_score_)

best_clf = random_search.best_estimator_

# Training set performance
y_train_pred = best_clf.predict(X_train)
train_acc_score = accuracy_score(y_train, y_train_pred)
print("Training accuracy score with best parameters:", train_acc_score)

# Test set performance
y_test_pred = best_clf.predict(X_test)
test_acc_score = accuracy_score(y_test, y_test_pred)
print("Testing accuracy score with best parameters:", test_acc_score)

# training set
print("Confusion matrix (training set):")
print(confusion_matrix(y_train, y_train_pred))
print("Classification report (training set):")
print(classification_report(y_train, y_train_pred))

# test set
print("Confusion matrix (test set):")
print(confusion_matrix(y_test, y_test_pred))
print("Classification report (test set):")
print(classification_report(y_test, y_test_pred))
```

```
       False       1.00      1.00      1.00       204
        True       1.00      1.00      1.00      7796

    accuracy                           1.00      8000
   macro avg       1.00      1.00      1.00      8000
weighted avg       1.00      1.00      1.00      8000

Confusion matrix (test set):
[[  33   28]
 [   1 1938]]
Classification report (test set):
              precision    recall  f1-score   support

       False       0.97      0.54      0.69        61
        True       0.99      1.00      0.99      1939

    accuracy                           0.99      2000
   macro avg       0.98      0.77      0.84      2000
weighted avg       0.99      0.99      0.98      2000
```

Based on these results, it appears that my model is overall performing well however, there is an imbalance of True and False with my Recall which is also affecting my f1 score. It looks like my model does a poor job of predicting false instances. My random forest results are

Best parameters found: {'n_estimators': 500, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 20, 'bootstrap': False}. However, I am going to add one more hyperparameter class_weight = to add more weight to the false category to see if this helps

```
class_weights = {False: 10, True: 1}

# Train a Random Forest classifier
clf = RandomForestClassifier(n_estimators= 500,
                             min_samples_split= 2,
                             min_samples_leaf= 2,
                             max_features= 'auto',
                             max_depth= 20,
                             bootstrap= False,
                             random_state=2,
                             class_weight= class_weights
                             )
clf.fit(X_train, y_train)

# Predict on the training set w/ accuracy score
y_train_pred = clf.predict(X_train)
train_acc_score = accuracy_score(y_train, y_train_pred)
print("Training accuracy score:", train_acc_score)
```

```python
# Predict on the test set w/ accuracy score
y_test_pred = clf.predict(X_test)
test_acc_score = accuracy_score(y_test, y_test_pred)
print("Testing accuracy score:", test_acc_score)

# training set
print("Confusion matrix (training set):")
print(confusion_matrix(y_train, y_train_pred))
print("Classification report (training set):")
print(classification_report(y_train, y_train_pred))

# test set
print("Confusion matrix (test set):")
print(confusion_matrix(y_test, y_test_pred))
print("Classification report (test set):")
print(classification_report(y_test, y_test_pred))
```

```
/shared-libs/python3.9/py/lib/python3.9/site-packages/sklearn/ensemble/_forest.py:427: FutureWarning: `max
  warn(
Training accuracy score: 0.999875
Testing accuracy score: 0.9895
Confusion matrix (training set):
[[ 204    0]
 [   1 7795]]
Classification report (training set):
              precision    recall  f1-score   support

       False       1.00      1.00      1.00       204
        True       1.00      1.00      1.00      7796

    accuracy                           1.00      8000
   macro avg       1.00      1.00      1.00      8000
weighted avg       1.00      1.00      1.00      8000

Confusion matrix (test set):
[[  44   17]
 [   4 1935]]
Classification report (test set):
              precision    recall  f1-score   support

       False       0.92      0.72      0.81        61
        True       0.99      1.00      0.99      1939

    accuracy                           0.99      2000
   macro avg       0.95      0.86      0.90      2000
weighted avg       0.99      0.99      0.99      2000
```

After adding this change I noticed an improvement in my macro f1 score changing from .84 to .90 in my testing set. Since my recall in the false category changed from .54 to .72 I am very happy with this change.

Regression (30 points)

placement (10 points)

Choose the best **ONE** attribute from 'academic', 'campus', and 'internship' for predicting how much the annual income a student may have using **simple linear regression**.

I am going to train 3 separate simple linear regression models for each attribute listed above and evaluate their performance using R-squared. I will also consider MSE and if this value is way too large compared to the others then that could indicate it may not be the best choice. I will not do any hyperparameter tuning because this is a simple linear regression.

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error

# Assuming df is your DataFrame
features = ['academic', 'campus', 'internship']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[features], df['annual'], test_size=0.

best_feature = None
best_r2_score = -100
best_mse = float('inf')

for feature in features:
    # simple linear regression model on each feature
    X_train_feat = X_train[[feature]]
    X_test_feat = X_test[[feature]]

    model = LinearRegression()
    model.fit(X_train_feat, y_train)

    # Predict on the test set w/ R-squared and mean squared error
    y_test_pred = model.predict(X_test_feat)
    r2 = r2_score(y_test, y_test_pred)
    mse = mean_squared_error(y_test, y_test_pred)

    print(f"{feature} R-squared: {r2:.4f}, Mean squared error: {mse:.4f}")

    if r2 > best_r2_score:
        best_r2_score = r2
        best_mse = mse
        best_feature = feature
```

```
print(f"Best attribute for predicting annual income: {best_feature}")
```

```
academic R-squared: 0.5070, Mean squared error: 1051643468.4340
campus R-squared: 0.2845, Mean squared error: 1526286338.9922
internship R-squared: 0.4200, Mean squared error: 1237209002.9527
Best attribute for predicting annual income: academic
```

placement (10 points)

Choose the best **ONE** attribute from 'academic', 'campus', and 'internship' for predicting how much the annual income a student may have using **simple polynomial regression** with **the regularization terms** of your choice.

For this question, I am going to be using The ridge method. I think Ridge will work well because I think academic, campus, and internship are correlated values. Also, since this is a simple polynomial regression I will not be tuning hyperparameters

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score, mean_squared_error


features = ['academic', 'campus', 'internship']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[features], df['annual'], test_size=0.


poly = PolynomialFeatures(2)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

best_feature = None
best_r2_score = -100
best_mse = float('inf')

# regularization parameter
alpha = 1.0

for i, feature in enumerate(features):
    # Train a Ridge regression model on each feature
    X_train_feat = X_train_poly[:, [i+1]]
    X_test_feat = X_test_poly[:, [i+1]]

    model = Ridge(alpha=alpha, random_state=2)
    model.fit(X_train_feat, y_train)
```

```
    # Prediction on the test set w/ R-squared and mean squared error
    y_test_pred = model.predict(X_test_feat)
    r2 = r2_score(y_test, y_test_pred)
    mse = mean_squared_error(y_test, y_test_pred)

    print(f"{feature} R-squared: {r2:.4f}, Mean squared error: {mse:.4f}")

    if r2 > best_r2_score:
        best_r2_score = r2
        best_mse = mse
        best_feature = feature

print(f"Best attribute for predicting annual income: {best_feature}")
```

```
academic R-squared: 0.5070, Mean squared error: 1051643105.3685
campus R-squared: 0.2845, Mean squared error: 1526286264.0071
internship R-squared: 0.4200, Mean squared error: 1237208674.2806
Best attribute for predicting annual income: academic
```

placement (10 points)

Use 'academic', 'campus', and 'internship' together for predicting how much the annual income a student may have using **multiple linear regression**.

```
features = ['academic', 'campus', 'internship']

X_train, X_test, y_train, y_test = train_test_split(df[features], df['annual'], test_size=0.

# multiple linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

y_test_pred = model.predict(X_test)

# R-squared and mean squared error
r2 = r2_score(y_test, y_test_pred)
mse = mean_squared_error(y_test, y_test_pred)

print(f"R-squared: {r2:.3f}, Mean squared error: {mse:.2f}")
```

```
R-squared: 0.862, Mean squared error: 294739859.23
```

```
import numpy as np
rmse = np.sqrt(mse)
```

```
print(f"Root Mean Squared Error: {rmse:.2f}")
```

```
Root Mean Squared Error: 17167.99
```

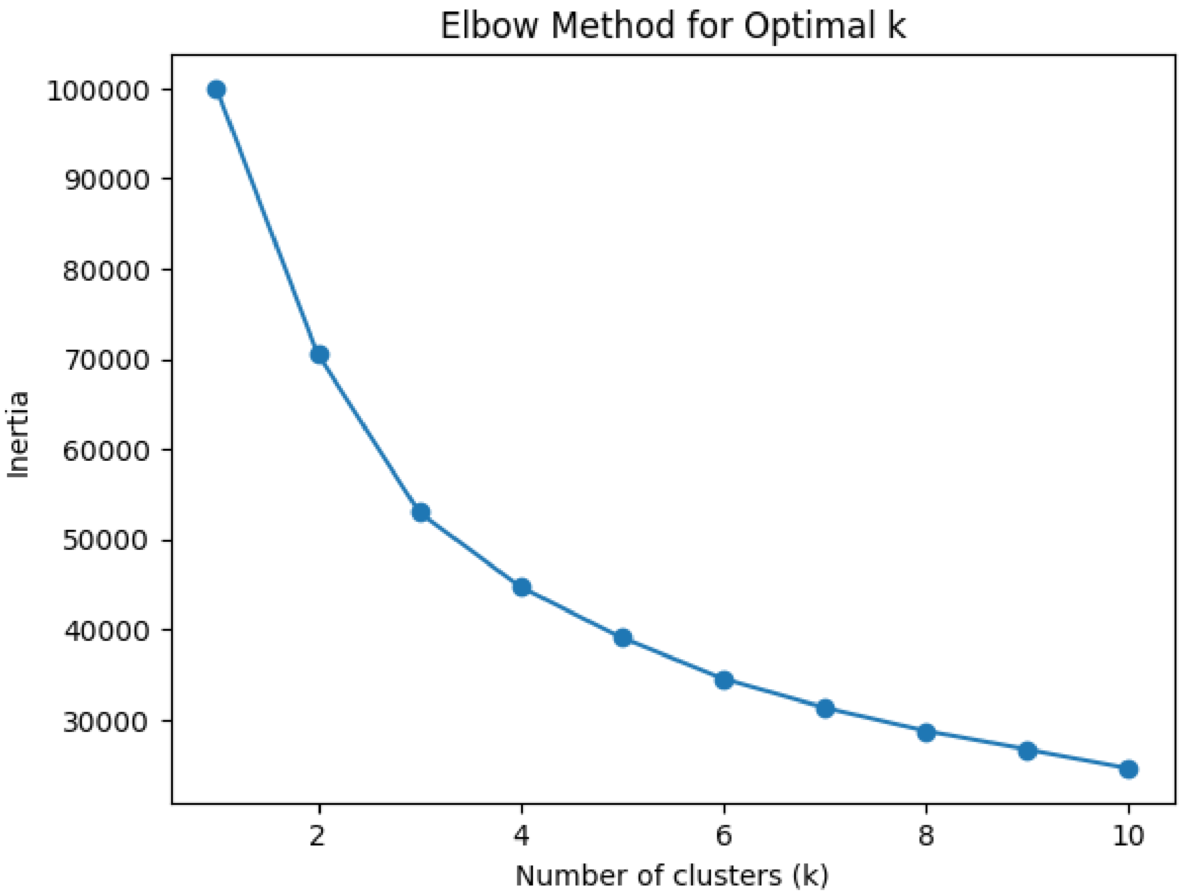
It looks like 86% of the variance can be explained by the variables academic, campus, and internship. I could not interpret the MSE values so I decided to convert them to the RMSE to see the average difference between predicted and actual values. $17,167 as the error is pretty good but not great.

Clustering (20 points)

K-Means (5 points)

Use 'c01', 'c02', ..., 'c10' and K-Means, cluster similar students to *k* groups. Explain why you choose such a *k*.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

features = ['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10']

# scale
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[features])

inertias = []
k_values = list(range(1, 11))

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=2)
    kmeans.fit(scaled_features)
    inertias.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(k_values, inertias, marker='o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```




```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[features])

# Find the best value of k using silhouette score
best_score = -1
best_k = 0
for k in range(2, 10):
    kmeans = KMeans(n_clusters=k, random_state=2)
    kmeans.fit(X_scaled)
    score = silhouette_score(X_scaled, kmeans.labels_)
    if score > best_score:
        best_score = score
        best_k = k

print(f"Best value of k: {best_k}, Best silhouette score: {best_score}")
```

```
Best value of k: 3, Best silhouette score: 0.2893368088403269
```

```
k = 3
kmeans = KMeans(n_clusters=k, random_state=2)
```

```
kmeans.fit(scaled_features)

# cluster labels
df['cluster'] = kmeans.labels_
```

```
df.head()
```

	Unnamed: 0 int64	id int64	major object	gender object	c01 float64	c02
0	0	0	Computer Science	M	75.75757575757575	82
1	1	1	Electric Engineering	M	81.81818181818181	90
2	2	2	Computer Science	M	81.21212121212122	83
3	3	3	Computer Science	F	82.42424242424242	85.
4	4	4	Computer Science	M	86.66666666666667	88

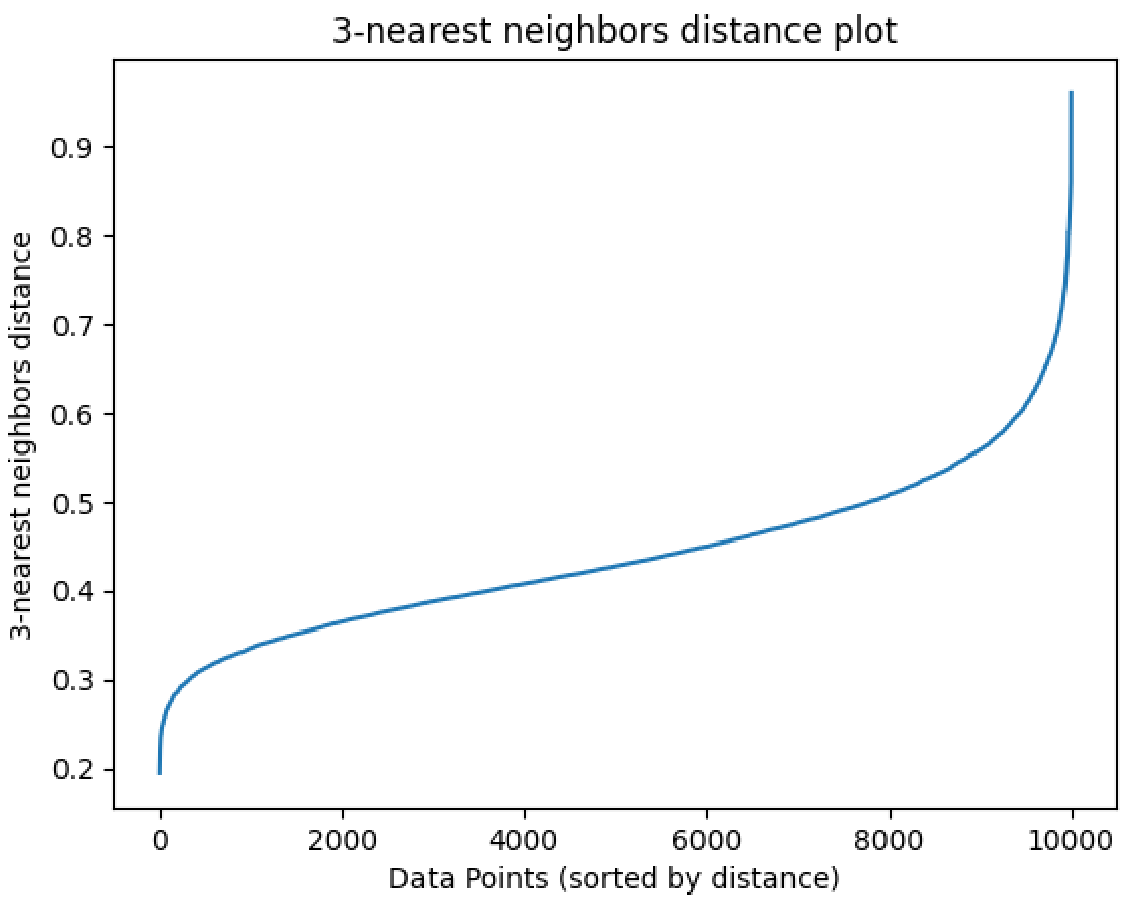
I chose a K of 3 based on the elbow method and the silhouette score. There seems to be a sharp slope change at 3 and the best silhouette score is also at 3.

DBSCAN (5 points)

Use 'c01', 'c02', ..., 'c10' and DBSCAN, cluster similar students to groups. Explain why you choose such epsilon and minpoints.

```
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt

features = ['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10']

# Scale
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[features])

k = 3
nN = NearestNeighbors(n_neighbors=k)
nN.fit(scaled_features)
distances, _ = nN.kneighbors(scaled_features)
distances = np.sort(distances[:, -1])
```

```python
plt.plot(distances)
plt.xlabel('Data Points (sorted by distance)')
plt.ylabel('3-nearest neighbors distance')
plt.title('3-nearest neighbors distance plot')
plt.show()
```




```python
#based on the KNN distance plot
epsilon = 0.6
min_samp = 4

# DBSCAN clustering
dbscan = DBSCAN(eps=epsilon, min_samples=min_samp)
dbscan.fit(scaled_features)

# Assign the cluster labels to each data point
df['cluster'] = dbscan.labels_
```

```python
df.head(15)
```

	Unnamed: 0 int64	id int64	major object	gender object	c01 float64	c02
	0 - 14	0 - 14	Computer Sc... 40% Electric Engi... 40%	M 80%	71.51515151515152 ...	77.3

			2 others 20%	F 20%		
0	0	0	Computer Science	M	75.75757575757575	82
1	1	1	Electric Engineering	M	81.81818181818181	90
2	2	2	Computer Science	M	81.21212121212122	83
3	3	3	Computer Science	F	82.42424242424242	85.
4	4	4	Computer Science	M	86.66666666666667	88
5	5	5	Information Science	M	76.96969696969697	87.
6	6	6	Electric Engineering	M	85.45454545454545	93
7	7	7	Electric Engineering	M	80.0	90
8	8	8	Electric Engineering	F	85.45454545454545	90
9	9	9	Computer Science	M	75.75757575757575	77.

```
df['cluster'].value_counts()
```

```
 26     459
-1      440
 13     244
 33     177
 21     135
       ...
 222      4
 335      4
 316      3
 255      3
 340      3
Name: cluster, Length: 343, dtype: int64
```

I decided to use an eps of .6 based on the distance plot I created as that is where there was a large curve. For min_samples, I experimented with a couple of different values but ended up using 4 because it limited the number of values it considered as "noise" but is not so small that it clusters everything.

PCA (5 points)

Use PCA methods to reduce columns 'c01', 'c02', ..., 'c10','academic', 'campus', and 'internship' to **3 extracted attributes**. Print out the % of explained variance with the 3 extracted features.

```
from sklearn.decomposition import PCA

features = features = ['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10']

scaler = StandardScaler()
standardized_features = scaler.fit_transform(df[features])

pca = PCA(n_components=3)
principal_components = pca.fit_transform(standardized_features)

# percentage of explained variance
print("Explained variance ratio:", pca.explained_variance_ratio_)
```

```
Explained variance ratio: [0.45819083 0.28278882 0.12340587]
```

this output represents the percentage of variance explained by each of the 3 variables. It looks like academic has the most with 45%, campus is next with 28%, and internship is last with 12%. Similar to my findings with the multiple linear regression, these 3 variables make up 86% of the total variance in the data.

PCA (5 points)

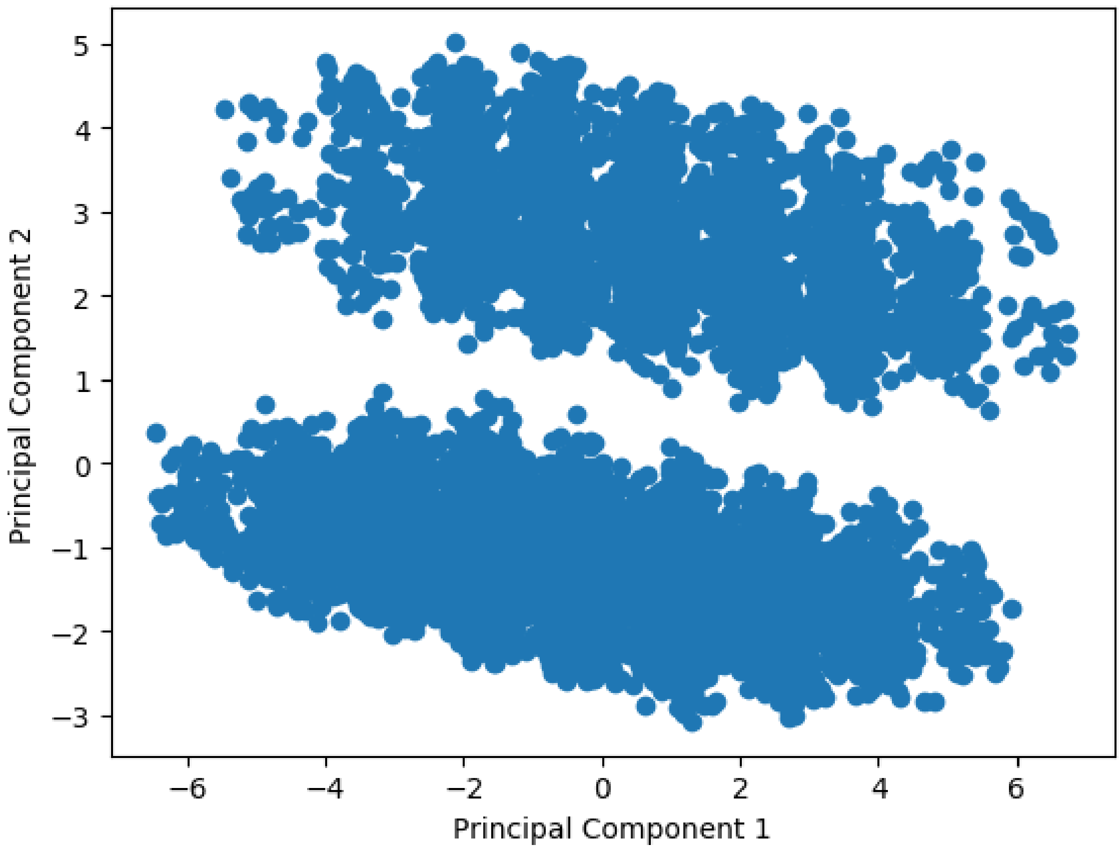
Use PCA methods to reduce columns 'c01', 'c02', ..., 'c10','academic', 'campus', and 'internship' to **80% explained variance**. Print out the number of extracted features we need.

```
features = features = ['c01', 'c02', 'c03', 'c04', 'c05', 'c06', 'c07', 'c08', 'c09', 'c10']
scaler = StandardScaler()
standardized_features = scaler.fit_transform(df[features])

# 80% of explained variance
pca = PCA(n_components=0.8, random_state=2)
principal_components = pca.fit_transform(standardized_features)
print(principal_components)
# extracted features
print("Number of extracted features:", pca.n_components_)
```

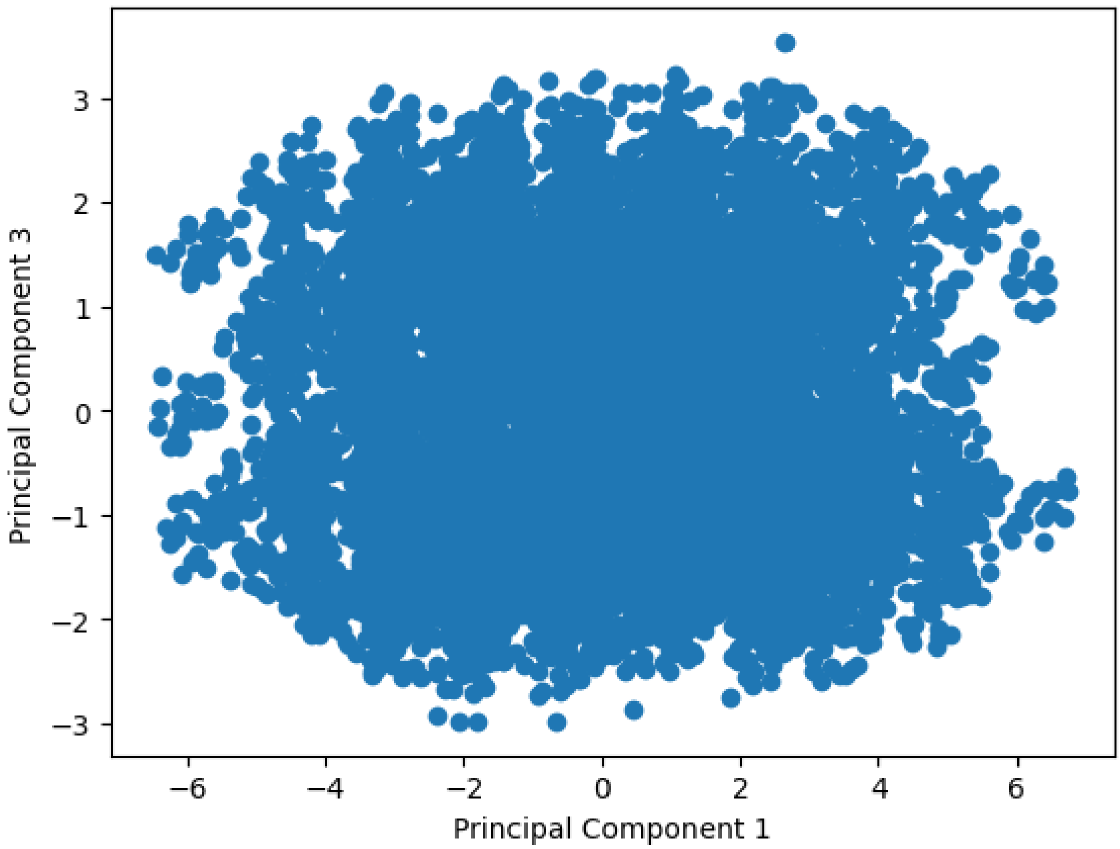
```
[[-1.61823721 -0.4196714   0.20382262]
 [ 1.37409966 -0.16808357  2.30708668]
 [-0.704156   -2.28882722 -1.17526   ]
 ...
 [ 3.29727541  2.70135797  0.28720139]
 [-0.39492525 -1.74009281 -0.73209071]
```

```
 [ 3.87350844  1.56838084 -1.52530017]]
Number of extracted features: 3
```

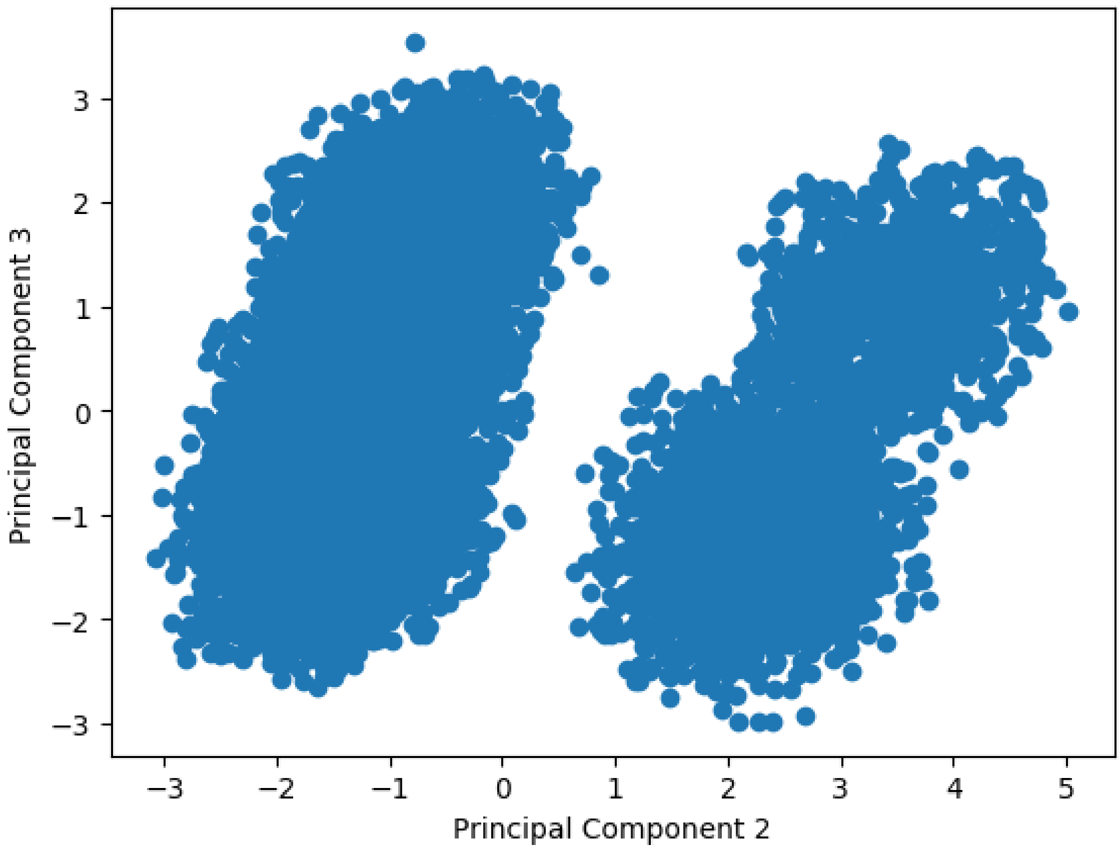
```python
import matplotlib.pyplot as plt

plt.scatter(principal_components[:, 0], principal_components[:, 1])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```




```python
plt.scatter(principal_components[:, 0], principal_components[:, 2])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 3')
plt.show()
```




```
plt.scatter(principal_components[:, 1], principal_components[:, 2])
plt.xlabel('Principal Component 2')
plt.ylabel('Principal Component 3')
plt.show()
```



this further confirms that 3 variables 'academic', 'campus', and 'internship' explain 80% of the variance

Frequent Patterns (10 points)

There is an attribute 'elective'. Use *apriori* algorithm, to find the frequent patterns of the elective courses and find the association rules. As the hypterparameter training, you should set the min_support, min_confidence, and min_lift so that your conclusion is interesting.

```
! pip install apyori
! pip install mlxtend
```

```
Requirement already satisfied: apyori in /root/venv/lib/python3.9/site-packages (1.1.2)
WARNING: You are using pip version 22.0.4; however, version 23.1 is available.
You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command.
Requirement already satisfied: mlxtend in /root/venv/lib/python3.9/site-packages (0.22.0)
Requirement already satisfied: matplotlib>=3.0.0 in /shared-libs/python3.9/py/lib/python3.9/site-packages
Requirement already satisfied: pandas>=0.24.2 in /shared-libs/python3.9/py/lib/python3.9/site-packages (fr
Requirement already satisfied: scipy>=1.2.1 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from
Requirement already satisfied: joblib>=0.13.2 in /shared-libs/python3.9/py/lib/python3.9/site-packages (fr
Requirement already satisfied: scikit-learn>=1.0.2 in /shared-libs/python3.9/py/lib/python3.9/site-package
```

```
Requirement already satisfied: numpy>=1.16.2 in /shared-libs/python3.9/py/lib/python3.9/site-packages (fron
Requirement already satisfied: setuptools in /root/venv/lib/python3.9/site-packages (from mlxtend) (58.1.0)
Requirement already satisfied: pyparsing>=2.2.1 in /shared-libs/python3.9/py-core/lib/python3.9/site-packag
Requirement already satisfied: contourpy>=1.0.1 in /shared-libs/python3.9/py/lib/python3.9/site-packages (
Requirement already satisfied: packaging>=20.0 in /shared-libs/python3.9/py-core/lib/python3.9/site-package
Requirement already satisfied: pillow>=6.2.0 in /shared-libs/python3.9/py/lib/python3.9/site-packages (fron
Requirement already satisfied: python-dateutil>=2.7 in /shared-libs/python3.9/py-core/lib/python3.9/site-pa
Requirement already satisfied: fonttools>=4.22.0 in /shared-libs/python3.9/py/lib/python3.9/site-packages
Requirement already satisfied: kiwisolver>=1.0.1 in /shared-libs/python3.9/py/lib/python3.9/site-packages
Requirement already satisfied: cycler>=0.10 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from
Requirement already satisfied: pytz>=2017.3 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from
Requirement already satisfied: threadpoolctl>=2.0.0 in /shared-libs/python3.9/py/lib/python3.9/site-package
Requirement already satisfied: six>=1.5 in /shared-libs/python3.9/py-core/lib/python3.9/site-packages (fron
WARNING: You are using pip version 22.0.4; however, version 23.1 is available.
You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command.
```

```
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
from apyori import apriori
```

```
df['elective'] = df['elective'].str.replace('\n', '').str.strip().str.split(' ')
```

```
df['elective'].head()
```

```
0    [['Databases', 'Statistical_Inference', 'High_...
1    [['Databases', 'Data_Mining', 'Big_Data', 'Nat...
2    [['Text_Marketing_Analysis', 'Databases', 'Sta...
3    [['Effective_Communication', 'Data_Structures_...
4    [['R_for_Data_Science', 'Databases', 'Machine_...
Name: elective, dtype: object
```

For this section, I found that the hyperparameter that affect my results the most was my min_support and min lift. with a min support of .05, this means itemsets need to appear in at least 5% of the datasets. with a min lift of 1.2, this means that there must be a positive association between the words selected. I think it's interesting that "data mining" is frequently chosen!

```
rules_2_itemset = apriori(df['elective'], min_support=0.05, min_confidence=0.1, min_lift=1.2

# Display rules
count = 0
for rule in list(rules_2_itemset):
    print(rule)
```

```
    count +=1
print("Rule Count:", count)
```

```
RelationRecord(items=frozenset({"'Machine_Learning']", "'Data_Mining'", "'Text_Marketing_Analysis'"}), sup
RelationRecord(items=frozenset({"'Machine_Learning']", "'Data_Mining'", "'Vital_Skills_for_Data_Scientists
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Data_Structures_and_Algorithms'", "'Data_Sc
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Databases'", "'Data_Structures_and_Algorith
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Datacenter_Computing'", "'Data_Structures_a
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Data_Structures_and_Algorithms'", "'Deep_Le
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Effective_Communication'", "'Data_Structure
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'High_Performance_and_Parallel_Computing'",
RelationRecord(items=frozenset({"'Machine_Learning'", "'Python_for_Data_Science'", "'Data_Structures_and_A
RelationRecord(items=frozenset({"'Machine_Learning']", "'Python_for_Data_Science'", "'Data_Structures_and_
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Data_Structures_and_Algorithms'", "'Nature_
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Neural_Networks'", "'Data_Structures_and_Al
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Data_Structures_and_Algorithms'", "'R_for_D
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Data_Structures_and_Algorithms'", "'Statist
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Data_Structures_and_Algorithms'", "'Statist
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Statistical_Modeling'", "'Data_Structures_a
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Stats_for_Data_Science'", "'Data_Structures
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Data_Structures_and_Algorithms'", "'Text_Ma
RelationRecord(items=frozenset({"'Python_for_Data_Science'", "'Vital_Skills_for_Data_Scientists'", "'Data_
RelationRecord(items=frozenset({"'Machine_Learning']", "'Big_Data'", "'Python_for_Data_Science'", "'Data_M
RelationRecord(items=frozenset({"'Machine_Learning']", "'Data_Mining'", "'Python_for_Data_Science'", "'Dat
RelationRecord(items=frozenset({"'Machine_Learning']", "'Data_Mining'", "'Data_Structures_and_Algorithms'"
RelationRecord(items=frozenset({"'Machine_Learning']", "'Databases'", "'Data_Mining'", "'Python_for_Data_S
RelationRecord(items=frozenset({"'Machine_Learning']", "'Datacenter_Computing'", "'Data_Mining'", "'Python
RelationRecord(items=frozenset({"'Machine_Learning']", "'Data_Mining'", "'Python_for_Data_Science'", "'Eff
RelationRecord(items=frozenset({"'Machine_Learning']", "'Data_Mining'", "'Python_for_Data_Science'", "'Sta
RelationRecord(items=frozenset({"'Machine_Learning']", "'Stats_for_Data_Science'", "'Data_Mining'", "'Pyth
RelationRecord(items=frozenset({"'Machine_Learning']", "'Data_Mining'", "'Python_for_Data_Science'", "'Tex
RelationRecord(items=frozenset({"'Machine_Learning']", "'Data_Mining'", "'Vital_Skills_for_Data_Scientists
Rule Count: 49
```

Neural Network (*Optional*)

Choose one or two attributes you like, train a simple and a deep neural network. Compare the result and summarize your experience.