Question 1:

Part A:

1. Support is the frequency with which an item set appears in the transactions in the data set. The formula to calculate it is I/T where I is the number of transactions containing the item set and T is the total number of transactions. For an association rule, the item set in question is that which contains the items on the left and right side of the rule.

   Confidence is the probability that an item set contains the item on the right side of an association rule given that it contains the item on the left side. The formula to calculate it is P(A and B) / P(A) where the association rule is A -> B. This can also be calculated as support({A,B})/support({A}).

   Lift is a measure of how much more likely an item is to appear in a transaction given that another item appears. The formula to calculate it is P(A union B)/(P(A)*P(B)) where A and B are the events that a certain item appears in a transaction.

2. The apriori algorithm works by starting with all item sets of size 1, checking each to see if they meet the minimum support threshold, then checking item sets of size 2 only if all item sets of size 1 that constitute that item set meet the minimum support threshold. This process repeats for item sets of size 3, 4, 5 and so on. The pruning of item sets to check that this algorithm executes, called apriori pruning, is valid because of the property that if an item is infrequent, all of its supersets must also be infrequent. This pruning is important because for very large databases of transactions, checking all item sets for the minimum support threshold is extremely computationally expensive. In many cases, apriori pruning can dramatically reduce the number of item sets to be checked because the number of item sets pruned grows significantly as the number of items in the set increases.

3. support({Bread, Milk}) = P(Bread and Milk) = ⅗ = 0.6
   confidence(bread -> milk) = support({Bread, Milk}) / support({Bread}) = 0.6/0.8 = 0.75
   This item set meets both the minimum support and confidence thresholds.

Part B:

Python Code:

```python
from itertools import combinations

products = ["Laptop", "Phone", "Headphones", "Charger", "Mouse",
"Keyboard", "Monitor", "Tablet", "Smartwatch", "Speakers"]
# Fixed set of transactions for deterministic results
```

```
transactions = [
["Laptop", "Phone", "Charger"],
["Phone", "Headphones", "Charger", "Smartwatch"],
["Laptop", "Tablet", "Speakers"],
["Monitor", "Keyboard", "Mouse"],
["Laptop", "Monitor", "Keyboard", "Mouse"],
["Phone", "Smartwatch"],
["Headphones", "Charger"],
["Laptop", "Headphones", "Charger"],
["Monitor", "Tablet", "Speakers"],
["Laptop", "Phone", "Headphones", "Smartwatch"],
["Laptop", "Phone", "Mouse", "Keyboard"],
] * 91 # Repeat the fixed set 91 times to have 1001 transactions

def get_item_sets(products, k):
    item_sets = list(combinations(products, k))
    return [list(item_set) for item_set in item_sets]

def check_support(item_sets, min_support):
    supported_item_sets = []
    for item_set in item_sets:
        support = 0
        for transaction in transactions:
            if(all(item in transaction for item in item_set)):
                support = support + 1
        support = support / len(transactions)
        if(support >= min_support):
            supported_item_sets.append(item_set)

    return supported_item_sets

def apriori(transactions, products, min_support):
    supported_item_sets = []
    candidates = get_item_sets(products,1)
    max_k = max(len(sublist) for sublist in transactions)
    for k in range(1,max_k+1):
        #Add supported item sets to result
        supported_sets_k = check_support(candidates,min_support)
        for item in supported_sets_k:
            supported_item_sets.append(item)

        #get candidates for next iteration
        candidates = []
        sets_kplus1 = get_item_sets(products, k+1)
```

```
        for item_set in sets_kplus1:
            subsets = get_item_sets(item_set,k)
            if all(subset in supported_sets_k for subset in subsets):
                candidates.append(item_set)

    return supported_item_sets

print(apriori(transactions,products,0.185))
```

1. [['Laptop'], ['Phone'], ['Headphones'], ['Charger'], ['Mouse'], ['Keyboard'], ['Monitor'], ['Smartwatch'], ['Laptop', 'Phone'], ['Phone', 'Smartwatch'], ['Headphones', 'Charger'], ['Mouse', 'Keyboard']]

2. The only item sets where the size is greater than 1 that meet the minimum support threshold are ['Laptop', 'Phone'], ['Phone', 'Smartwatch'], ['Headphones', 'Charger'], and ['Mouse', 'Keyboard']. The possible association rules from these are ['Laptop' -> 'Phone'], ['Phone' -> 'Laptop'], ['Phone' -> 'Smartwatch'], ['Smartwatch' -> 'Phone'], ['Headphones' -> 'Charger'], ['Charger' -> 'Headphones'], ['Mouse' -> 'Keyboard'], ['Keyboard' -> 'Mouse']. The top 5 of these rules in lift that meet the minimum confidence of 0.6 are ['Mouse' -> 'Keyboard'] (Lift = 4.33), ['Keyboard' -> 'Mouse'] (Lift = 4.33), ['Phone' -> 'Smartwatch'] (Lift = 2.6), ['Smartwatch' -> 'Phone'] (Lift = 2.6), ['Headphones' -> 'Charger'] (Lift = 2.44).

Question 2:

Part A:

1. Row ID 2 is missing a weight value. This can be replaced by the average weight value, 71.25.
   Row ID 4 is missing an age value. This can be replaced by the average age value, 31.75.
   Row ID 5 is missing a height value. This can be replaced by the average height value, 171.75.

   The resulting data set is:

ID	Age	Weight	Height	Blood Pressure
1	25	68	172	120
2	42	71.25	160	140

3	31	72	175	117
4	31.75	80	180	300
5	29	65	171.75	122

2. In the blood pressure column, the median is 122 and the IQR is 120-140. 300 is over 1.5 times the length of the IQR over the high end of the IQR, so it's an outlier. This outlier will be capped at a value 1.5 times the length of the IQR over the high end of the IQR, which in this case is 170.

   The resulting data set is:

ID	Age	Weight	Height	Blood Pressure
1	25	68	172	120
2	42	71.25	160	140
3	31	72	175	117
4	31.75	80	180	170
5	29	65	171.75	122

3. There are no categorical columns in this data set.

   The resulting data set is:

ID	Age	Weight	Height	Blood Pressure
1	25	68	172	120
2	42	71.25	160	140
3	31	72	175	117
4	31.75	80	180	170
5	29	65	171.75	122

Part B:

1. After normalizing the age, weight and height columns using min-max normalization, the resulting data set is:

ID	Age	Weight	Height	Blood Pressure
1	0	0.2	0.6	120
2	1	0.417	0	140
3	0.353	0.467	0.75	117
4	0.397	1	1	170
5	0.235	0	0.563	122

2. After normalizing the blood pressure column using z-score normalization, the resulting data set is:

ID	Age	Weight	Height	Blood Pressure
1	0	0.2	0.6	-0.623
2	1	0.417	0	0.28
3	0.353	0.467	0.75	-0.759
4	0.397	1	1	1.635
5	0.235	0	0.563	-0.533

3. Min-max normalization doesn't change the distribution of the data, rather it simply scales it down to fit between 0 and 1. Z score normalization fits the data to a normal distribution with the mean and standard deviation calculated from the data. Min-max normalization is better to use when the data is not normally distributed, while z score normalization is better to use when the data is normally distributed and there are significant outliers.

Question 3:

Part A:

1. K fold cross validation is a technique to evaluate the performance of a machine learning model. As opposed to a traditional train test split where one portion of the data is used to

train the model and the other is used to test its accuracy, k fold cross validation splits the data into k equally sized portions and trains a model k times on all the data not in the given fold then tests it on the data in that fold. This approach for model evaluation is beneficial because it avoids the arbitrary nature of having some data be used for training and some for testing by having all of the data be used for training and testing the same number of times. This can lower variance and allow models to generalize better.

2. 178 rows of data divided equally into 5 folds would result in 35.6 rows per fold. In practice, three of the folds contain 36 rows of data and the other two contain 35. 5 fold cross validation will train a model on all data not in fold 1, test that model on fold 1, train a model on all data not in fold 2, test that model on fold 2, train a model on all data not in fold 3, test that model on fold 3, train a model on all data not in fold 4, test that model on fold 4, train a model on all data not in fold 5, test that model on fold 5, then take the average of whatever evaluation metric was used to evaluate the test sets to get the final evaluation of the model. The following Python code splits the wine data set into 5 folds and displays the training and testing indices for each fold.

```python

import pandas as pd
from sklearn.model_selection import KFold
import numpy as np

df = pd.read_csv('wine.data', header=None)
column_names = ['class','Alcohol','Malic acid','Ash','Alcalinity of ash','Magnesium','Total
phenols','Flavanoids','Nonflavanoid phenols','Proanthocyanins','Color
intensity','Hue','OD280/OD315 of diluted wines','Proline']
df.columns = column_names

kfold = KFold(n_splits=5, shuffle=True, random_state=42)

fold_indices = [(train_index, test_index) for train_index, test_index in kfold.split(df)]

for i, (train_index, test_index) in enumerate(fold_indices):
    print(f"Fold {i+1}:")
    print("Training indices:", train_index)
    print("Testing indices:", test_index)
    print(len(test_index))

```

3. The advantage of using a larger k in k fold cross validation is that the model will be trained and tested more times overall thus it will be less subject to high variance. The advantage of using a smaller k is that it’s less computationally expensive because you

don't have to train the model as many times and each of the test sets are larger so you're not subject to a biased evaluation of each model due to a small test set.

Part B:

1. True Positives for vineyard 1: 10
   True Negatives for vineyard 1: 27
   False Positives for vineyard 1: 1
   False Negatives for vineyard 1: 2
   Accuracy = correct/total = 37/40 = 0.925
   Precision = true positives / (true positives + false positives) = 10/11 = 0.909
   Recall = true positives / (true positives + false negatives) = 10/12 = 0.833
   F1 Score = 2*((precision*recall)/(precision+recall)) = 2*((0.909*0.833)/(0.909+0.833)) = 0.870

   Accuracy is an overall metric for the classification model that is all encompassing but isn't always the best metric especially if there's high class imbalance as there is here (a lot more wines aren't vineyard 1 than are). Precision represents the model's relative ability to avoid misclassifying wines that truly aren't from vineyard 1. Recall represents the model's relative ability to avoid misclassifying wines that truly are from vineyard 1. F1 score is a combination of precision and recall that may be used as an overall evaluation metric like accuracy and it may be more informative in the case of high class imbalance.

2. Using accuracy alone to evaluate a classification model is bad in the case of class imbalance because it rewards improved predictive ability in the most common class more than the other ones. In the case of the wine model where we are evaluating whether the wine was from vineyard 1 as in part 1, there is much more data in the non vineyard 1 class than in the vineyard 1 class, so it suffers from this pitfall of accuracy. Considering precision and recall also offers insight into the tradeoff between predicting actual positives correctly and predicting actual negatives correctly based on your classification threshold that accuracy doesn't offer. If you are considering where to set the classification threshold you will want to see how that affects precision and recall to find the optimal value.

3. In this context, the distributor should prioritize recall over other metrics. This is because recall is the metric that is directly determined by the number of false negatives given a certain number of true positives.

Question 4:

Part A:

1. Info(Data) = I(3,3) = -0.5*log_2(0.5) - 0.5*log_2(0.5) = 1
   Info_Time_of_day(Data) = 1/3 * I(1,1) + 1/3 * I(1,1) + 1/6 * I(1,0) + 1/6 * I(0,1) = ⅔ * (-0.5*log_2(0.5) - 0.5*log_2(0.5)) = 0.667
   Info_Device(Data) = 1/2 * I(1,2) + 1/3 * I(2,0) + 1/6 * I(0,1) = 1/2 * (-0.333333*log_2(0.333333) - 0.666667*log_2(0.666667)) = 0.459
   Info_Category(Data) = 1/3 * I(2,0) + 1/3 * I(2,0) + 1/6 * I(1,0) + 1/6 * I(0,1) = 0

   Based on the data provided, I would choose 'Category' as the root node of the decision tree because it has the highest information gain (1-0) of all the features. The shopper is most likely to make a purchase if the category is clothing or groceries and not if the category is electronics or books.

2. Assuming the decision tree only allows binary splits, the depth 1 tree would look like this:

   Category is Clothing?

No: Purchase = No	Yes: Purchase = Yes

   If the decision tree algorithm allows for multiway splits (some do but the standard one does not) then the tree would look like this:

   Category is Clothing or Groceries?

No: Purchase = No	Yes: Purchase = Yes

   In the data set, the shopper is most likely to make a purchase if the category is clothing or groceries and not if the category is electronics or books. Clothing is the chosen split if the tree does not allow for multiway splits because it results in the highest information gain.

Part B:

1. Info(Data) = I(3,3) = -0.5*log_2(0.5) - 0.5*log_2(0.5) = 1

2. Info_Time_of_day(Data) = 1/3 * I(1,1) + 1/3 * I(1,1) + 1/6 * I(1,0) + 1/6 * I(0,1) = ⅔ * (-0.5*log_2(0.5) - 0.5*log_2(0.5)) = 0.667
   IG(Time_of_day) = 1-0.667 = 0.333
   Info_Device(Data) = 1/2 * I(1,2) + 1/3 * I(2,0) + 1/6 * I(0,1) = 1/2 * (-0.333333*log_2(0.333333) - 0.666667*log_2(0.666667)) = 0.459
   IG(Device) = 1-0.459 = 0.541

Info_Category(Data) = 1/3 * I(2,0) + 1/3 * I(2,0) + 1/6 * I(1,0) + 1/6 * I(0,1) = 0
IG(Category) = 1-0 = 1

Category has the highest information gain so it is used as the root node of the decision tree.

3. Information gain is calculated by computing the measure of information, entropy or gini index for example, of the response variable if split on a certain variable and subtracting it by the information of the response variable before the split. Information gain is critical in the process of creating a decision tree because it's the criteria for how the tree is built. Each node is created by splitting on the feature with the highest information gain.