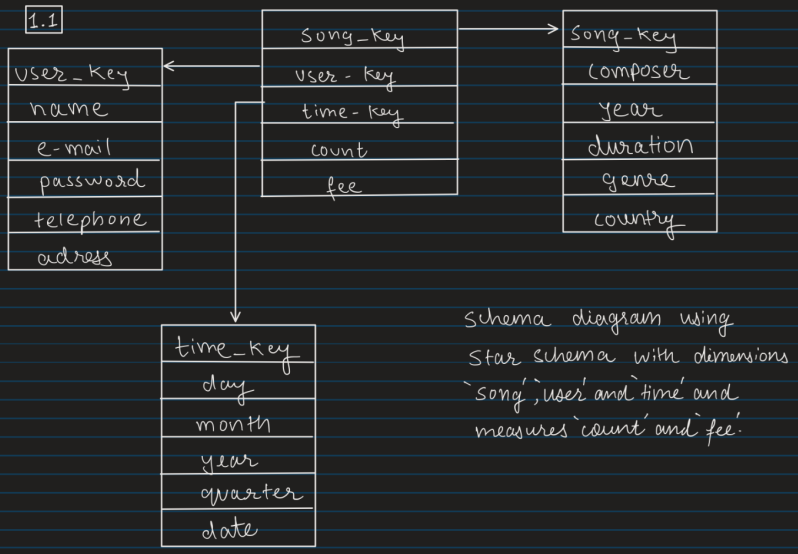
DATA MINING ASSIGNMENT 2 [PART 1]

Question 1

Q1

1.1



Schema diagram using Star schema with dimensions 'song', 'user' and 'time' and measures 'count' and 'fee'.

1.2

To list the total fee collected for a given song for a given month of a given year we will have to use two OLAP operations. First 'Roll-up' and then 'dice'. Roll up is used to summarise the data by climbing up the hierarchy like we are supposed to do with the dimension 'time'. We go up from 'month' to 'year'. Now the second operation 'dice' is used to select 'two' or more dimensions from the data like we are supposed to do for the dimensions 'time' and 'song'.

1.3

Assuming time dimensions has '4' levels (day, month, quarter, year) while song and user have 1 level.

The total number of cuboids that can be in the cube are :-

n = dimensions

$$T = \prod_{i=1}^{n} (L_i + 1) \qquad L_i = \text{no. of levels associated with level } i.$$

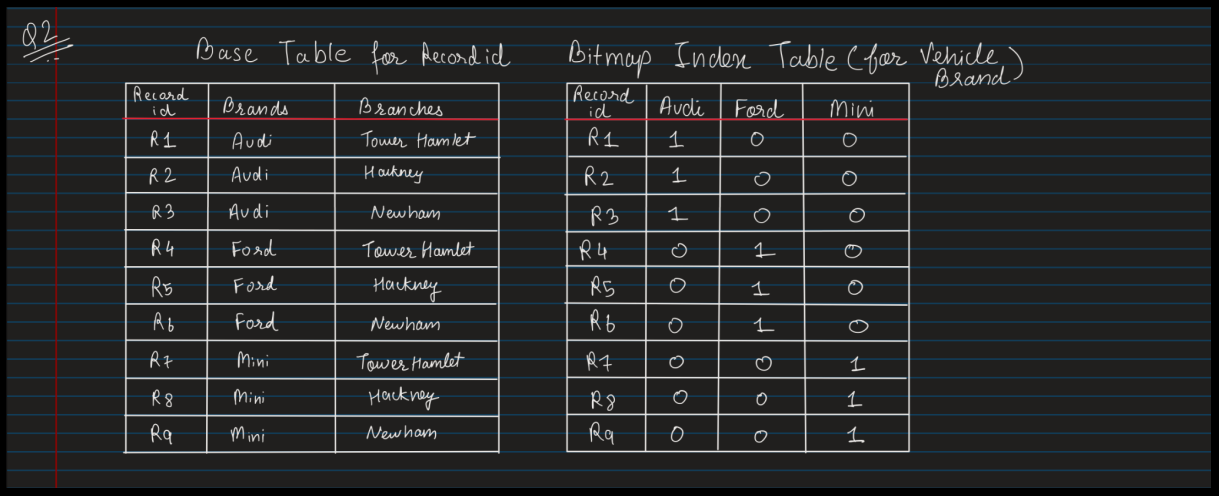
$$\therefore T = (4+1)_{time} \times (1+1)_{song} \times (1\times 1)_{user}$$

$$T = (5) \times (2) \times (2)$$

$$= \boxed{20}$$

The given cube can contain 20 cuboids.

Question 2



Record id	Brands	Branches
R1	Audi	Tower Hamlet
R2	Audi	Hackney
R3	Audi	Newham
R4	Ford	Tower Hamlet
R5	Ford	Hackney
R6	Ford	Newham
R7	Mini	Tower Hamlet
R8	Mini	Hackney
R9	Mini	Newham

Record id	Audi	Ford	Mini
R1	1	0	0
R2	1	0	0
R3	1	0	0
R4	0	1	0
R5	0	1	0
R6	0	1	0
R7	0	0	1
R8	0	0	1
R9	0	0	1

Question 3

Adding minimum and maximum aggregate measures to the json file and then producing the values per year which is 2009 and 2010.

importing the necessary libraries

```
from sqlalchemy import create_engine
from cubes.tutorial.sql import create_table_from_csv
```

creating a table 'ibrd_balance' using the given csv file

```
[31] engine = create_engine('sqlite:///data.sqlite')
     create_table_from_csv(engine,
                           "IBRD_Balance_Sheet__FY2010.csv",
                           table_name="ibrd_balance",
                           fields=[
                               ("category", "string"),
                               ("category_label", "string"),
                               ("subcategory", "string"),
                               ("subcategory_label", "string"),
                               ("line_item", "string"),
                               ("year", "integer"),
                               ("amount", "integer")],
                           create_id=True
                          )
```

```
[33] workspace.import_model("tutorial_model.json")
```

creating a data cube and loading our table into it

```
[34] cube = workspace.cube("ibrd_balance")
```

creating an instance of browser which will further be used to perform aggregation on our data cube

```
[35] browser = workspace.browser(cube)
```

After adding the min_fun and max_fun measures to the json file we use the drilldown function to produce the minimum and maximum values of 'record_count' and 'amount_sum'

```
result = browser.aggregate()
result.summary["record_count"]
```

```
result.summary["amount_sum"]
```

```
result = browser.aggregate(drilldown=["year"])
for record in result:
    print(record)
```

output

```
{'year': 2009, 'amount_sum': 550840, 'min_fun': -1683, 'max_fun': 110040, 'record_count': 31}
{'year': 2010, 'amount_sum': 566020, 'min_fun': -3043, 'max_fun': 128577, 'record_count': 31}
```

The modified json file

```
{
    "dimensions": [
        {
         "name":"item",
         "levels": [
                    {
                        "name":"category",
                        "label":"Category",
                        "attributes": ["category", "category_label"]
                    },
                    {
                        "name":"subcategory",
                        "label":"Sub-category",
                        "attributes": ["subcategory", "subcategory_label"]
                    },
                    {
                        "name":"line_item",
                        "label":"Line Item",
                        "attributes": ["line_item"]
                    }
                ]
        },
        {"name":"year", "role": "time"}
    ],
    "cubes": [
        {
            "name": "ibrd_balance",
            "dimensions": ["item", "year"],
            "measures": [{"name":"amount", "label":"Amount"}],
            "aggregates": [
                        {
                            "name": "amount_sum",
                            "function": "sum",
                            "measure": "amount"
```

Adding the aggregate measures max_fun and min_fun

```
                    {
                      "name": "min_fun",
                      "function": "min",
                      "measure": "amount"
                    },

                    {
                      "name": "max_fun",
                      "function": "max",
                      "measure": "amount"

                    },

                    {
                        "name": "record_count",
                        "function": "count"
                    }
                ],
            "mappings": {
                            "item.line_item": "line_item",
                            "item.subcategory": "subcategory",
                            "item.subcategory_label": "subcategory_label",
                            "item.category": "category",
                            "item.category_label": "category_label"
                           },
            "info": {
                "min_date": "2010-01-01",
                "max_date": "2010-12-31"
            }
        }
    ]
```

Question 4

1) *Modifying the json file as per the requirements (adding dimension region,age,online_shopper)*

```
{
    "dimensions": [
        {"name":"region"},
        {"name":"age"},
        {"name":"online_shopper"}
    ],
    "cubes": [
        {
            "name": "country-income",
            "dimensions": ["region", "age","online_shopper"],
            "measures": [{"name":"income", "label":"income"}],
            "aggregates": [
                    {
                        "name": "amount_sum",
                        "function": "sum",
                        "measure": "income"
                    },
                    {
                      "name": "min_fun",
                      "function": "min",
                      "measure": "income"
                    },

                    {
                      "name": "max_fun",
                      "function": "max",
                      "measure": "income"

                    },

                    {
                        "name": "avg_fun",
                        "function": "avg",
                        "measure":"income"
                    }
                ]
        }
    ]
}
```

2) *Loading the new json file and the csv file, displaying the aggregate measures and then sorting them as per region and online shopper. Further displaying the aggregate measures for people between the age of 40 and 50 using the RangeCut Function*

```
[26] # creating a table from the csv file
     engine = create_engine('sqlite:///data.sqlite')
     create_table_from_csv(engine,
                           "country-income.csv",
                           table_name="country-income",
                           fields=[
                               ("region", "string"),
                               ("age", "integer"),
                               ("income", "integer"),
                               ("online_shopper", "string")],
                           create_id=True
                          )
```

```
[27] # creating an instancce of a cube
     from cubes import Workspace
     workspace = Workspace()
     workspace.register_default_store("sql", url="sqlite:///data.sqlite")
```

```
     # loading our table into the cube

     workspace.import_model('assignment.json')
     cube = workspace.cube("country-income")
     browser = workspace.browser(cube)
```

```
[30] # displaying the min,max and avg aggregates
     result = browser.aggregate()
     result.summary
```

```
     {'amount_sum': 768200, 'avg_fun': 76820.0, 'max_fun': 99600, 'min_fun': 57600}
```

```
     # displaying the required result as per the column region
     result = browser.aggregate(drilldown=["region"])
     for record in result:
         print(record)
```

```
     {'region': 'Brazil', 'amount_sum': 193200, 'min_fun': 57600, 'max_fun': 73200, 'avg_fun': 64400.0}
     {'region': 'India', 'amount_sum': 331200, 'min_fun': 69600, 'max_fun': 94800, 'avg_fun': 82800.0}
     {'region': 'USA', 'amount_sum': 243800, 'min_fun': 64800, 'max_fun': 99600, 'avg_fun': 81266.66666666667}
```

```
[32] # displaying the required result as per the column Online Shoppers
     result = browser.aggregate(drilldown=["online_shopper"])
     for record in result:
         print(record)
```

```
     {'online_shopper': 'No', 'amount_sum': 386400, 'min_fun': 62400, 'max_fun': 99600, 'avg_fun': 77280.0}
     {'online_shopper': 'Yes', 'amount_sum': 381800, 'min_fun': 57600, 'max_fun': 94800, 'avg_fun': 76360.0}
```

```
[34] # using the function RangeCut displaying the aggregate measures for people with age between 40 and 50
     import cubes as cubes
     cuts = [cubes.RangeCut("age", ["40"],["50"])]
     cell = cubes.Cell(cube, cuts)
     result = browser.aggregate(cell, drilldown=["age"])
     for record in result:
         print(record)
```

```
     {'age': 40, 'amount_sum': 69600, 'min_fun': 69600, 'max_fun': 69600, 'avg_fun': 69600.0}
     {'age': 42, 'amount_sum': 80400, 'min_fun': 80400, 'max_fun': 80400, 'avg_fun': 80400.0}
     {'age': 43, 'amount_sum': 73200, 'min_fun': 73200, 'max_fun': 73200, 'avg_fun': 73200.0}
     {'age': 45, 'amount_sum': 79400, 'min_fun': 79400, 'max_fun': 79400, 'avg_fun': 79400.0}
     {'age': 46, 'amount_sum': 62400, 'min_fun': 62400, 'max_fun': 62400, 'avg_fun': 62400.0}
     {'age': 49, 'amount_sum': 86400, 'min_fun': 86400, 'max_fun': 86400, 'avg_fun': 86400.0}
```

Summary

```
[18] # printing the summary of our result which contains the aggregates of our measures
     result.summary
```

```
     {'amount_sum': 451400,
      'avg_fun': 75233.33333333333,
      'max_fun': 86400,
      'min_fun': 62400}
```

DATA MINING ASSIGNMENT 2 [PART 2]

Question 1

Q1 Given $x_1 = (1, -1)$ $x_2 = (-1, 1)$

Class of the first observation $y_1 = 0$

Class of the second observation $y_2 = 1$

To classify the observation $x = (-2, 3)$

For 1-nearest neighbour we need to calculate the distance of the point x with x_1 and x_2, and then classify it to the class of the nearest of the two.

∴ distance from x_1

$d_{x_1} = \sqrt{(3-(-1))^2 + (-2-1)^2}$

$= \sqrt{16 + 9} = \sqrt{25}$

$d_{x_1} = 5$

distance from x_2.

$d_{x_2} = \sqrt{(3-1)^2 + (-2-(-1))^2}$

$= \sqrt{4 + 1}$

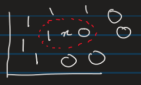
$d_{x_2} = \sqrt{5}$

Hence $d_{x_2} < d_{x_1}$.

∴ We will classify x to the class of x_2 which $y_2 = 1$

Question 2

Q2 When a new observation is to be classified to the dataset D, according to the K nearest neighbour classifier it checks its K nearest points and assigns the new observation to the class of points which are in majority. For example, a new observation x is to be classified into either 1 or 0.

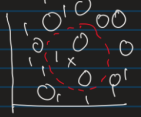


Consider even value for K.

K = 2

Hence x chooses its 2 closest points. One is from 0 and other 1. So here x needs a 'tie breaking policy' to be classified in either of the classes.

Now if K is an odd number (K = 3)



Here x has 3 closest neighbours, two 0's and one 1. Therefore it can easily be classified in to the class of the majority of the 3 points which is '0'.

Hence we do not require a tie breaker policy in K nearest neighbour, when the value of K is 'odd'.

Question 3

Q3

The accuracy of a classifier is evaluated by

$$\frac{\text{total number of correct predictions}}{\text{total number of instances}}.$$

Lets say, we have a dataset with 1000 samples.
999 samples belong to class 'A' and 1 sample belongs to class 'B'.

Now if we try to evaluate the accuracy of this classifier, we will end up with

$\frac{999}{1000} = 99.9 \cdot 1$. accuracy of predicting the class 'A'.

Such data sets when used to make predictions give extremely good accuracy for a particular class but very bad accuracy for the other. (Falsely classifying class B into class A most of the times).

Hence inspite of the fact that the classifier has a 99.1 accuracy, the classifier is said to perform poorly.

Therefore other metrics such as the precision and recall are considered to evaluate the performance of a classifier

Question 4

Q4

Positive class → y

Precision = 1.0

Recall = 0.1

If this classifier predicts that an observation does not belong to class y. Should it be trusted.

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive + False positive}} = 1.0 \quad -(1)$$

$\therefore$ True positive = True positive + False positive.

This implies that there are no false positives.

Now,

$$\text{Recall} = 0.1 = \frac{\text{True positive}}{\text{True positive + False negative}}. \quad -(2)$$

According to (1), there are no false positives, which means no observations were falsely classified as 'y'.

If we take into consideration the recall which is 'low' as compared to the precision., we can say that a lot of observations are falsely classified into the other class(negative class)

Hence there are high number of false negatives.

1) Should it be trusted when the classifier predicts that an observation does not belong to class 'y'.

Ans. No, as there are high number of 'false negatives', there is a high chance, that an observation is falsely classified into the negative class.

2) Should it be trusted when the classifier predicts that an observation belongs to class y.

Ans. Yes, as the value of precision is 1.0 and no observations are falsely classified into the positive class. There is a high chance that an observation is correctly classified into the positive class.

Question 5

Q5

From looking at the confusion matrix below we can say that most number of false positives occur with the pair (4, 9) and hence it can be said that it is the most confusing pair of all.

Confusion matrix:

	0	1	2	3	4	5	6	7	8	9
0	38	0	0	0	0	0	1	0	0	0
1	0	40	0	0	0	0	0	0	0	0
2	0	3	39	0	0	2	0	1	1	0
3	0	1	1	39	0	1	0	1	0	1
4	0	3	0	0	40	0	0	1	0	3
5	0	0	0	1	0	32	1	0	2	0
6	0	1	0	0	1	1	39	0	0	0
7	0	0	0	0	2	1	0	34	0	2
8	0	2	0	0	0	2	0	0	27	1
9	0	0	0	0	4	0	0	1	1	29

4 times '4' is falsely classified as '9' and 3 times '9' is falsely classified as '4'. The pair 4 and 9 has most number of misclassifications.

Question 6 (Code)

```
[66] # calling the library to perform support vector machine classification from sklearn
     from sklearn import svm

     # creating an SVM instance with the kernel set to linear
     sv_var = svm.SVC()

     # Fitting our training data to the svm classfier in order to train it
     sv_var.fit(X_train, y_train)

     # using our model to make predictions on the test data which is X_test
     y_pred = sv_var.predict(X_test)
```

```
[70] # calling the metrics library from sklearn in order to calculate the accuracy and model metrics
     from sklearn import metrics

     # displaying the model accuracy using the function accuracy_score from the library metrics
     print("Accuracy is",metrics.accuracy_score(y_test, y_pred)*100,'%')
```

Output

```
Accuracy is 92.75 %
```

```
# displaying the model precision using the function precision_score from the library metrics
print("Precision:",metrics.precision_score(y_test, y_pred, average=None))

# displaying the model recall using the function recall_score from the library metrics
print("Recall:",metrics.recall_score(y_test, y_pred, average=None))
```

Output

```
Precision: [0.975      0.95238095 0.93617021 0.97560976 0.84615385 0.86842105
 0.95       0.97058824 1.         0.83333333]
Recall: [1.         1.         0.95652174 0.90909091 0.93617021 0.91666667
 0.9047619  0.84615385 0.9375     0.85714286]
```

Question 7 (Code)

```
[55] from sklearn.ensemble import RandomForestClassifier
     from sklearn.model_selection import GridSearchCV
     # creating a the parameter grid as per the requirement
     param_grid = {
         'max_features': [0.1, 0.25],
         'n_estimators': [50, 100, 200]
     }
     # creating an instance of the random forest classifier to further classify our data
     rfc = RandomForestClassifier()
     # create the grid search model
     grid_search = GridSearchCV(rfc, param_grid,
                                cv = 5)
```

```
[56] # fitting our training data into the grid search model(random foresy classifier)
     grid_search.fit(X_train, y_train)
```

Output

```
{'max_features': 0.1, 'n_estimators': 200}
```

```
[61] # using the method best_params_ printing the best parameters computed from the algorithm
     best_p = grid_search.best_params_
     print('The best parameters computed after grid search are',best_p)
```

After performing the grid search for 5 fold cross validation using the Random Forest Classifier we obtained the following best parameters and obtained the accuracy of best model to be 91.5%

```
The best parameters computed after grid search are {'max_features': 0.1, 'n_estimators': 200}
```

```
[ ] # printing the accuracy of the model with the best parameters
    accuracy = grid_search.score(X_test, y_test)
    print('The accuacy of the best model on the test data is',accuracy*100,'%')
```

```
The accuacy of the best model on the test data is 91.5 %
```