Open in Colab

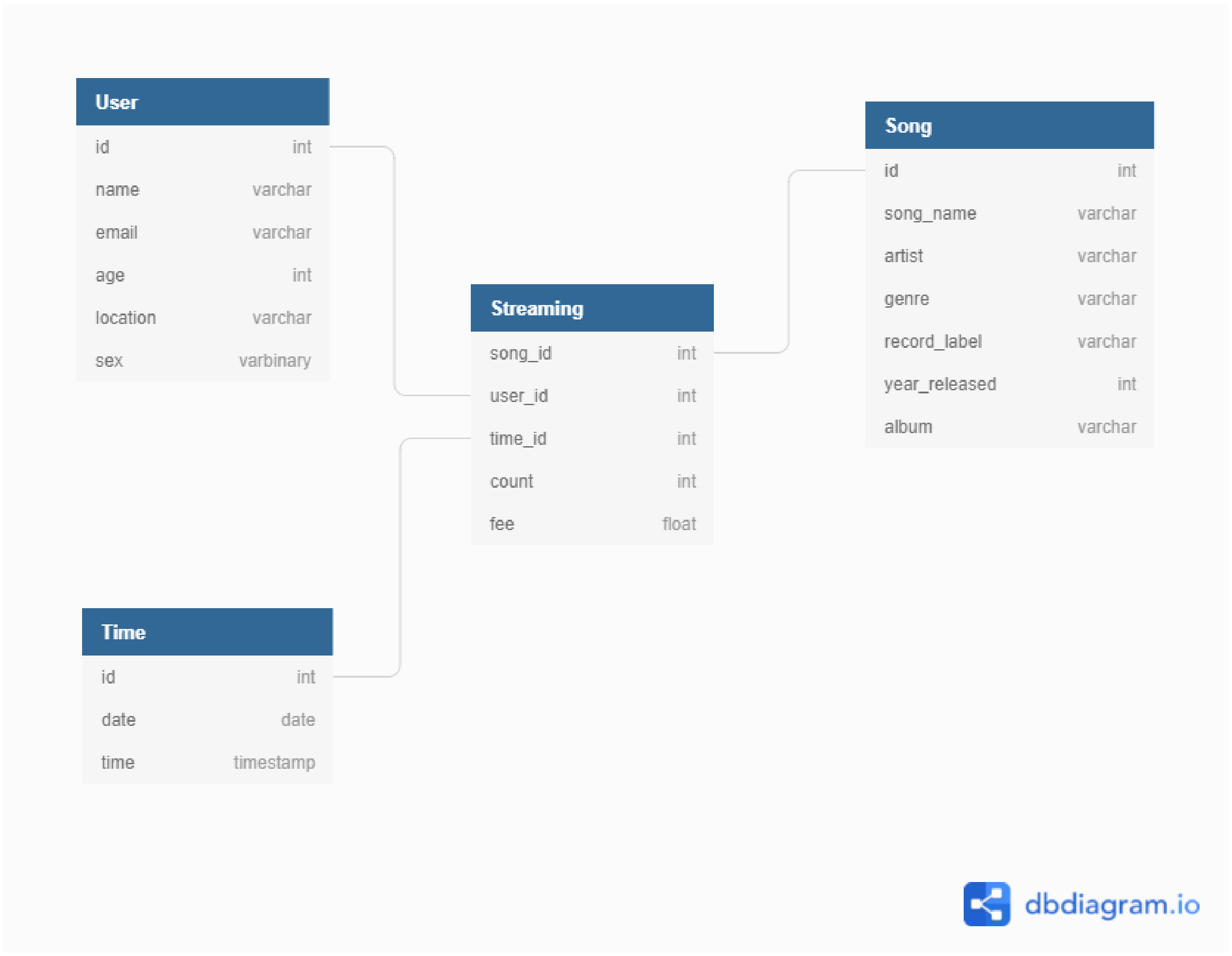
```
#!pip install cubes
#!pip install -Iv sqlalchemy==1.3.9
```

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

ECS766 Coursework 2 - Elliot Linsey

Q.1A

Untitled.png



Q.1B

We need to perform a dicing operation as you are acting on 2 dimensions. This involves slicing by the given time period you wanted (October 2021), you may need to drill down to get to this specific month. Then slice by song dimension to the given song. From here you would use an aggregate function to produce the total fee from within that month.

Q1.C

Using the formula $\Pi_{i=1}^{n}(L_i + 1)$ equates to 5x2x2 = 20. Therefore, this cube will contain 20 cuboids.

Q2

Base Table

RID	Brand	Branch
R1	Audi	TH
R2	Audi	N
R3	Audi	H
R4	Ford	TH
R5	Ford	N
R6	Ford	H
R7	Mini	TH
R8	Mini	N
R9	Mini	H

Bitmap Index Table on Brand

RID	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

Q3

```
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
                     )
```

```
from cubes import Workspace

workspace = Workspace()
workspace.register_default_store("sql", url="sqlite:///data.sqlite")
workspace.import_model("tutorial_model.json")
cube = workspace.cube("ibrd_balance")
browser = workspace.browser(cube)
result = browser.aggregate()
result.summary["record_count"]
```

```
     62
```

Below is my JSON Model file, the two added functions are named 'maximum' and 'minimum' in the 'aggregates' section.

```
# {
#     "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"
#                     },
#                     {
#                         "name": "record_count",
#                         "function": "count"
#                     },
#                       {
#                         "name": "maximum",
#                         "function": "max",
#                         "measure": "amount"
#                     },
#                       {
#                         "name": "minimum",
#                         "function": "min",
#                         "measure": "amount"
#                     }
#                 ],
#             "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"
#             }
#         }
#     ]
# }
```

```
result = browser.aggregate(drilldown=["year"])
for record in result:
    print(str(record['year']) + " Minimum: " + str(record['minimum']))
    print(str(record['year']) + " Maximum: " + str(record['maximum']))
```

```
2009 Minimum: -1683
2009 Maximum: 110040
2010 Minimum: -3043
2010 Maximum: 128577
```

Q4

```
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
                     )

workspace = Workspace()
workspace.register_default_store("sql", url="sqlite:///data.sqlite")
workspace.import_model("country_income_model.json")
cube = workspace.cube("country_income")
browser = workspace.browser(cube)
result = browser.aggregate()
result.summary["record_count"]
```

```
     10
```

Below is my JSON file representation for the model of this data cube.

```
# {
#     "dimensions": [
#         {
#           "name": "region",
#           "levels":[{"name":"region",
#                     "label":"Region",
#                     "attributes": ["region"]}]
#         },
#         {
#           "name": "age",
#           "levels":[{"name":"age",
#                     "label":"Age",
#                     "attributes": ["age"]}]
#         },
#         {
#           "name": "online_shopper",
#           "levels":[{"name":"online_shopper",
#                     "label":"Online Shopper",
#                     "attributes": ["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": "record_count",
#                           "function": "count"
#                       },
#                         {
#                           "name": "maximum",
#                           "function": "max",
#                           "measure": "income"
#                       },
#                         {
#                           "name": "minimum",
#                           "function": "min",
#                           "measure": "income"
#                       },
#                       {
#                           "name": "average",
#                           "function": "avg",
#                           "measure": "income"
#                       }
#                   ],
#             "mappings": {
#                             "region.region": "region",
#                             "age.age": "age",
#                             "online_shopper.online_shopper": "online_shopper"
#                            }
#         }
#     ]
# }



result = browser.aggregate()
print('Full Summary:')
print(result.summary)
print('Region Summary:')
result = browser.aggregate(drilldown=["region"])
for record in result:
    print(record)
print('Online Shopper Summary:')
result = browser.aggregate(drilldown=["online_shopper"])
for record in result:
    print(record)
```

```
Full Summary:
{'amount_sum': 768200, 'record_count': 10, 'maximum': 99600, 'minimum': 57600, 'aver
Region Summary:
{'region': 'Brazil', 'amount_sum': 193200, 'record_count': 3, 'maximum': 73200, 'min
{'region': 'India', 'amount_sum': 331200, 'record_count': 4, 'maximum': 94800, 'mini
{'region': 'USA', 'amount_sum': 243800, 'record_count': 3, 'maximum': 99600, 'minimu
Online Shopper Summary:
{'online_shopper': 'No', 'amount_sum': 386400, 'record_count': 5, 'maximum': 99600,
{'online_shopper': 'Yes', 'amount_sum': 381800, 'record_count': 5, 'maximum': 94800,
import cubes as cubes
cuts = [cubes.RangeCut("age",[40],[50])]
cell = cubes.Cell(cube, cuts)
result = browser.aggregate(cell, drilldown=["age"])
print('Ages between 40 and 50 Summary:')
for record in result:
    print(record)
```

```
Ages between 40 and 50 Summary:
{'age': 40, 'amount_sum': 69600, 'record_count': 1, 'maximum': 69600, 'minimum': 696
{'age': 42, 'amount_sum': 80400, 'record_count': 1, 'maximum': 80400, 'minimum': 804
{'age': 43, 'amount_sum': 73200, 'record_count': 1, 'maximum': 73200, 'minimum': 732
{'age': 45, 'amount_sum': 79400, 'record_count': 1, 'maximum': 79400, 'minimum': 794
{'age': 46, 'amount_sum': 62400, 'record_count': 1, 'maximum': 62400, 'minimum': 624
{'age': 49, 'amount_sum': 86400, 'record_count': 1, 'maximum': 86400, 'minimum': 864
```

Part 2

Q1

The formula for Euclidean distance is $d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$

With these x values below we find the distances between x1 and x2 to x3.

```
x1 = (1,-1)
x2 = (-1,1)
x3 = (-2,3)


dist1 = ((-2-1)**2 + (3--1)**2)**0.5
print("distance 1 = " + str(dist1))

dist2 = ((-2--1)**2 + (3-1)**2)**0.5
print("distance 2 = " + str(dist2))
```

```
distance 1 = 5.0
distance 2 = 2.23606797749979
```

It would classify x3 as class y2 as the euclidean distance is closest to point x2 at 2.24 compared to x1 at 5.0.

Q2

When K is odd in a KNN algorithm we do not need a tie-breaking policy because one class will always have a greater number of points closer to the new point than the other class. For example if K=5, the closest you can get to a tie will be 3 points of class y1 and 2 points of class y2 being the nearest neighbours, in this situation and with all odd K numbers the point will always be designated as the class with the highest number of neighbours, therefore it would be classed as label y1.

Q3

A classifier that has an accuracy of 99.9% can be terrible for some datasets if 0.1% of the data is another class. This algorithm will label every data point as the same class and therefore miss the 0.1% that is different. While this appears to give a very high accuracy, it is actually not truly classifying anything rather than just assigning the same class to all data points. This can be extremely bad if misidentifying that 0.1% of data results in a far greater cost than identifying 99.9% of the other data.

Q4

A precision of 1.0 and low recall of 0.1 means that out of the data you predicted positive, all were correctly identified as true positive. However, the low recall means that you actually missed a lot of the true positives and mislabelled them as false negatives. Therefore, when it detects a positive it can be trusted but it is far less trustworthy when detecting negatives as there is a high chance they could be positive. In terms of this classifier, it should not be trusted if it detects that point to not belong to class y and it should be trusted if it detects it to belong to class y.

Q5

The K=1 classifier struggled the most with classes 4 and 9, correctly identifying 40 4s but misidentifying 4 of them (10%) as 9s.

Q6

```
import gzip
import pickle

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')

# Selecting the training data from the original dataset
f = gzip.open('mnist.pkl.gz', 'rb')
X, y = pickle.load(f, encoding='latin1')[0]
```

```
f.close()

# Subsampling
sample_size = 2000
X, y = X[:sample_size], y[:sample_size]
```

```
from sklearn.model_selection import train_test_split

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

```
from sklearn import svm
clf = svm.SVC()
clf.fit(X_train,y_train)
print('Test dataset accuracy: ' + str(clf.score(X_test,y_test)))
```

```
     Test dataset accuracy: 0.9275
```

Q7

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

parameters = {'n_estimators': [50,100,200], 'max_features': [0.1,0.25]}

rfc = RandomForestClassifier()
rfc_cv = GridSearchCV(rfc, parameters, cv=5)
rfc_cv.fit(X_train, y_train)
print('Best hyperparameter settings: {0}.'.format(rfc_cv.best_params_))
print('Average accuracy across folds of best hyperparameter setting: {0}.'.format(rfc_cv.b
print('Test dataset accuracy of best hyperparameter setting: {0}.'.format(rfc_cv.score(X_t
```

```
     Best hyperparameter settings: {'max_features': 0.1, 'n_estimators': 100}.
     Average accuracy across folds of best hyperparameter setting: 0.911875.
     Test dataset accuracy of best hyperparameter setting: 0.9025.
```

✓ 0s completed at 10:12 PM