

Customer De-identification Project

```
[1]: # Title: Data Privacy Certification Report
    ## Author: Scott Eugley
    ## Date: 10/20/2023

[2]: # Load in data and libraries

import pandas as pd
import numpy as np

raw_data = pd.read_excel("/Users/seugley/Desktop/GitHub/Data_De-identification/
↳Customer_Survey_Data.xlsx")

[3]: # Narrow down raw data by selecting the columns of interest: CustomerID, Gender,
    ↳Age, HouseholdIncome, and CardSpendMonthly

columns_of_interest = ['CustomerID', 'Gender', 'Age', 'HouseholdIncome',
↳'CardSpendMonthly']
data_of_interest = raw_data[columns_of_interest]

[4]: # === Data Quality Control ===

# First start by looking at the min and max values for each of the variables to
↳see if they make sense

# min and max values for age
min_age = data_of_interest['Age'].min()
max_age = data_of_interest['Age'].max()

# 10 min and 10 max values for household income
top_10_min_income = data_of_interest['HouseholdIncome'].nsmallest(10)
top_10_max_income = data_of_interest['HouseholdIncome'].nlargest(10)

# 10 min and 10 max values for 'card spend monthly
top_10_min_card_spend = data_of_interest['CardSpendMonthly'].nsmallest(10)
top_10_max_card_spend = data_of_interest['CardSpendMonthly'].nlargest(10)

# Print results
```

```

print("Minimum Age:", min_age)
print("Maximum Age:", max_age)

print("\nTop 10 Minimum Household Incomes:")
print(top_10_min_income)
print("\nTop 10 Maximum Household Incomes:")
print(top_10_max_income)

print("\nTop 10 Minimum Card Spending Monthly:")
print(top_10_min_card_spend)
print("\nTop 10 Maximum Card Spending Monthly:")
print(top_10_max_card_spend)

# There are some extreme high and low values, however, nothing appears to be out
↳ of the question. Some examples: a negative household income, an unusually high
↳ max age, a negative card spend monthly

# Identify any null, NA, or missing data points

columns_to_check = ['Gender', 'Age', 'HouseholdIncome', 'CardSpendMonthly']
missing_values = data_of_interest[columns_to_check].isnull().sum()

# Print results
print("Missing values in each column:")
print(missing_values)

# There are no missing values in the columns of interest

# Remove CustomerID column as this has no utility passed this point

columns_of_interest = ['Gender', 'Age', 'HouseholdIncome', 'CardSpendMonthly']
data_of_interest = raw_data[columns_of_interest]

# Change 0 and 1 to male and female respectively as establishing gender makes
↳ this data much more useful to the buyer. Made a copy of dataset to avoid
↳ warning message
data_of_interest = data_of_interest.copy()
data_of_interest.loc[:, 'Gender'] = data_of_interest['Gender'].map({0: 'Male', 1:
↳ 'Female'})

```

Minimum Age: 18
Maximum Age: 79

Top 10 Minimum Household Incomes:

135	9
311	9
321	9

338	9
427	9
510	9
600	9
604	9
655	9
666	9

Name: HouseholdIncome, dtype: int64

Top 10 Maximum Household Incomes:

1102	1073
2192	995
3068	780
3212	642
4949	575
3623	526
754	515
2061	472
4916	437
17	424

Name: HouseholdIncome, dtype: int64

Top 10 Minimum Card Spending Monthly:

1657	0.00
1716	0.00
2799	0.00
2878	0.00
4099	0.00
4714	0.00
4890	0.00
4025	6.97
3304	7.34
3549	7.53

Name: CardSpendMonthly, dtype: float64

Top 10 Maximum Card Spending Monthly:

3386	3926.41
1523	3104.63
1102	2969.39
2598	2503.25
1298	2461.03
508	1978.12
206	1899.93
2966	1894.91
2430	1799.19
711	1753.69

Name: CardSpendMonthly, dtype: float64

Missing values in each column:

```
Gender          0
Age             0
HouseholdIncome 0
CardSpendMonthly 0
dtype: int64
```

```
[5]: # === Creation of Equivalence Classes ===

# Establish age ranges
age_bins = [18, 35, 55, float('inf')]
age_labels = ['18-35', '36-55', '56+']

# Create an age group column
data_of_interest['AgeGroup'] = pd.cut(data_of_interest['Age'], bins=age_bins,
    ↳labels=age_labels)

# Define the income ranges
income_bins = [0, 36, 61, 101, float('inf')]
income_labels = ['0-35', '36-60', '61-100', '100+']

# Create a column for income group
data_of_interest['IncomeGroup'] = pd.cut(data_of_interest['HouseholdIncome'],
    ↳bins=income_bins, labels=income_labels)

# Define the card monthly spending groups
card_spend_bins = [0, 251, 601, float('inf')]
card_spend_labels = ['0-250', '251-600', '601+']

# Create a column for the card monthly spending group
data_of_interest['CardSpendGroup'] = pd.
    ↳cut(data_of_interest['CardSpendMonthly'], bins=card_spend_bins,
    ↳labels=card_spend_labels)

# Group by all variables to create equivalence classes
equivalence_classes = data_of_interest.groupby(['Gender', 'AgeGroup',
    ↳'IncomeGroup', 'CardSpendGroup'])

grouped_count = equivalence_classes.size()

unique_equivalence_classes = equivalence_classes.groups

unique_equivalence_classes = list(equivalence_classes.groups.keys())

# Calculate the count of records in each equivalence class and create a count
    ↳column
grouped_count = equivalence_classes.size().reset_index(name='count')
```

```

# Create a dataset of the equivalence classes
equivalence_classes_table = pd.DataFrame(unique_equivalence_classes,
    columns=['Gender', 'AgeGroup', 'IncomeGroup', 'CardSpendGroup'])

# Merge the equivalence classes dataset with the counts
equivalence_classes_with_count = equivalence_classes_table.merge(grouped_count,
    on=['Gender', 'AgeGroup', 'IncomeGroup', 'CardSpendGroup'])

# Print table with all rows
pd.set_option('display.max_rows', None)
print(equivalence_classes_with_count)

```

	Gender	AgeGroup	IncomeGroup	CardSpendGroup	count
0	Female	18-35	0-35	0-250	309
1	Female	18-35	0-35	251-600	220
2	Female	18-35	0-35	601+	25
3	Female	18-35	100+	0-250	3
4	Female	18-35	100+	251-600	7
5	Female	18-35	100+	601+	3
6	Female	18-35	36-60	0-250	70
7	Female	18-35	36-60	251-600	85
8	Female	18-35	36-60	601+	15
9	Female	18-35	61-100	0-250	21
10	Female	18-35	61-100	251-600	29
11	Female	18-35	61-100	601+	5
12	Female	36-55	0-35	0-250	97
13	Female	36-55	0-35	251-600	93
14	Female	36-55	0-35	601+	9
15	Female	36-55	100+	0-250	25
16	Female	36-55	100+	251-600	56
17	Female	36-55	100+	601+	20
18	Female	36-55	36-60	0-250	120
19	Female	36-55	36-60	251-600	126
20	Female	36-55	36-60	601+	26
21	Female	36-55	61-100	0-250	64
22	Female	36-55	61-100	251-600	119
23	Female	36-55	61-100	601+	38
24	Female	56+	0-35	0-250	275
25	Female	56+	0-35	251-600	113
26	Female	56+	0-35	601+	9
27	Female	56+	100+	0-250	27
28	Female	56+	100+	251-600	91
29	Female	56+	100+	601+	46
30	Female	56+	36-60	0-250	78
31	Female	56+	36-60	251-600	78
32	Female	56+	36-60	601+	14
33	Female	56+	61-100	0-250	47

34	Female	56+	61-100	251-600	89
35	Female	56+	61-100	601+	22
36	Male	18-35	0-35	0-250	255
37	Male	18-35	0-35	251-600	220
38	Male	18-35	0-35	601+	31
39	Male	18-35	100+	0-250	4
40	Male	18-35	100+	251-600	9
41	Male	18-35	100+	601+	5
42	Male	18-35	36-60	0-250	57
43	Male	18-35	36-60	251-600	68
44	Male	18-35	36-60	601+	21
45	Male	18-35	61-100	0-250	9
46	Male	18-35	61-100	251-600	26
47	Male	18-35	61-100	601+	11
48	Male	36-55	0-35	0-250	102
49	Male	36-55	0-35	251-600	99
50	Male	36-55	0-35	601+	8
51	Male	36-55	100+	0-250	26
52	Male	36-55	100+	251-600	69
53	Male	36-55	100+	601+	29
54	Male	36-55	36-60	0-250	107
55	Male	36-55	36-60	251-600	129
56	Male	36-55	36-60	601+	36
57	Male	36-55	61-100	0-250	54
58	Male	36-55	61-100	251-600	106
59	Male	36-55	61-100	601+	45
60	Male	56+	0-35	0-250	237
61	Male	56+	0-35	251-600	161
62	Male	56+	0-35	601+	11
63	Male	56+	100+	0-250	33
64	Male	56+	100+	251-600	76
65	Male	56+	100+	601+	45
66	Male	56+	36-60	0-250	70
67	Male	56+	36-60	251-600	86
68	Male	56+	36-60	601+	21
69	Male	56+	61-100	0-250	36
70	Male	56+	61-100	251-600	79
71	Male	56+	61-100	601+	32

```
[6]: # Check which ECs have the lowest counts

equivalence_classes_filtered = □
    ↳ equivalence_classes_with_count[equivalence_classes_with_count['count'] < 4]

print(equivalence_classes_filtered)
```

```
# The lowest count in an EC is 3, which implies an acceptable threshold risk of 33%
```

	Gender	AgeGroup	IncomeGroup	CardSpendGroup	count
3	Female	18-35	100+	0-250	3
5	Female	18-35	100+	601+	3

```
[7]: # === Assessment of Maximum and Median Risk ===

# Maximum risk is 33% due to smallest EC has 3 records

# Calculate reciprocal (1/count) and multiply by 100 of the count column to
# create a new column called risk%
equivalence_classes_with_count['risk%'] = (1 /
# equivalence_classes_with_count['count']) * 100

# Round risk% to two decimal places
equivalence_classes_with_count['risk%'] =
# equivalence_classes_with_count['risk%'].round(2)

# Median of the risk% column
median_risk = equivalence_classes_with_count['risk%'].median()

# Print median risk
print("Median Risk %:", median_risk)
print("Maximum Risk %:", 33.33)
```

Median Risk %: 2.1950000000000003

Maximum Risk %: 33.33

```
[8]: # === Scenario 1 ===

# In this scenario we consider the situation of rogue employees that are
# handling the data. We will assume that 100 people will be handling this data
# set

# In this example we will test a high risk (1/100) rogue employees, medium risk
# (5/10) rogue employees, and a conservative model (10/100) rogue employees

# Median risk (1/100)
median_risk_1 = (((0.02195) * (0.01))/0.01) * (0.01)

print("Pr(re-id) Median Risk (1/100):", median_risk_1)

# Maximum risk (1/100)
```

```

maximum_risk_1 = (((0.33) * (0.01))/0.01) * (0.01)

print("Pr(re-id) Maximum Risk (1/100):",maximum_risk_1)

# Median risk (5/100)

median_risk_2 = (((0.02195) * (0.05))/0.05) * (0.05)

print("Pr(re-id) Median Risk (5/100):",median_risk_2)

# Maximum risk (5/100)

maximum_risk_2 = (((0.33) * (0.05))/0.05) * (0.05)

print("Pr(re-id) Maximum Risk (5/100):",maximum_risk_2)

# Median risk (10/100)

median_risk_3 = (((0.02195) * (0.1))/0.1) * (0.1)

print("Pr(re-id) Median Risk (10/100):",median_risk_3)

# Maximum risk (10/100)

maximum_risk_3 = (((0.33) * (0.1))/0.1) * (0.1)

print("Pr(re-id) Maximum Risk (10/100):",maximum_risk_3)

```

```

Pr(re-id) Median Risk (1/100): 0.00021950000000000002
Pr(re-id) Maximum Risk (1/100): 0.00330000000000000004
Pr(re-id) Median Risk (5/100): 0.00109750000000000002
Pr(re-id) Maximum Risk (5/100): 0.0165
Pr(re-id) Median Risk (10/100): 0.00219500000000000003
Pr(re-id) Maximum Risk (10/100): 0.033

```

[9]: # === Scenario 2 ===

```

# In this scenario we consider an internal adversary where for example a data_
→analyst working with the data set recognizes a record belonging to someone in_
→their circle of acquaintances

# It is estimated that 97% of Americans have a cellphone plan. While this data_
→is being sold to a credit card company, this dataset contains individuals with_
→a phone plan, so that is the relevant prevalence to be used here

```



```

pr_acq = 1-(1-0.97)**150

# Median risk

pr_reid_median_risk = ((0.02195 * pr_acq) / pr_acq) * pr_acq

print("Pr(re-id) Median Risk:",pr_reid_median_risk)

# Maximum risk

pr_reid_maximum_risk = ((0.33 * pr_acq) / pr_acq) * pr_acq

print("Pr(re-id) Maximum Risk:",pr_reid_maximum_risk)

```

Pr(re-id) Median Risk: 0.02195

Pr(re-id) Maximum Risk: 0.33

```

[10]: # === Scenario 3 ===

# This scenario describes a situation in which a data breach occurs. Data is
→exposed outside of the intended recipients

# Median risk

pr_reid_median_risk_s3 = ((0.02195 * 0.27) / 0.27) * 0.27

# Maximum risk

pr_reid_maximum_risk_s3 = ((0.33 * 0.27) / 0.27) * 0.27

print("Pr(re-id) Median Risk:",pr_reid_median_risk_s3)

print("Pr(re-id) Maximum Risk:",pr_reid_maximum_risk_s3)

```

Pr(re-id) Median Risk: 0.0059265

Pr(re-id) Maximum Risk: 0.089100000000000001

```

[11]: # === Scenario 4 ===

# This scenario considers the most catastrophic situation, in which the dataset
→becomes exposed to the public

# Median risk

pr_reid_median_risk_s4 = ((0.02195 * 1) / 1) * 1

# Maximum risk

```

```

pr_reid_maximum_risk_s4 = ((0.33 * 1) / 1) * 1

print("Pr(re-id) Median Risk:",pr_reid_median_risk_s4)

print("Pr(re-id) Maximum Risk:",pr_reid_maximum_risk_s4)

```

Pr(re-id) Median Risk: 0.02195
Pr(re-id) Maximum Risk: 0.33

```

[12]: # === Results - Diagnostics Table ===

# Create column with probability of reidentification

equivalence_classes_with_count["pr_reid"] =
↳ equivalence_classes_with_count["risk%"] / 100

# Iterate scenario 1 equation through each EC and create a new column with the
↳ respective values

equivalence_classes_with_count["pr_reid_s1"] =
↳ ((equivalence_classes_with_count["pr_reid"] * 0.5) / 0.5) * 0.5

# Classify each value in the pr_reid_s1 into their respect percentage groups

equivalence_classes_with_count["risk%_group_s1"] = pd.cut(
    equivalence_classes_with_count["pr_reid_s1"],
    bins=[0, 0.05, 0.1, 0.2, 0.33, 0.5, float('inf')],
    labels=["<5%", "<10%", "<20%", "<33%", "<50%", ">50%"]
)

# Do the same with scenarion 2

equivalence_classes_with_count["pr_reid_s2"] =
↳ ((equivalence_classes_with_count["pr_reid"] * 1) / 1) * 1

equivalence_classes_with_count["risk%_group_s2"] = pd.cut(
    equivalence_classes_with_count["pr_reid_s2"],
    bins=[0, 0.05, 0.1, 0.2, 0.33, 0.5, float('inf')],
    labels=["<5%", "<10%", "<20%", "<33%", "<50%", ">50%"]
)

# Do the same with scenarion 3

equivalence_classes_with_count["pr_reid_s3"] =
↳ ((equivalence_classes_with_count["pr_reid"] * 0.27) / 0.27) * 0.27

```

```

equivalence_classes_with_count["risk%_group_s3"] = pd.cut(
    equivalence_classes_with_count["pr_reid_s3"],
    bins=[0, 0.05, 0.1, 0.2, 0.33, 0.5, float('inf')],
    labels=["<5%", "<10%", "<20%", "<33%", "<50%", ">50%"]
)

# Do the same with scenarion 4

equivalence_classes_with_count["pr_reid_s4"] =
    →((equivalence_classes_with_count["pr_reid"] * 1) / 1) * 1

equivalence_classes_with_count["risk%_group_s4"] = pd.cut(
    equivalence_classes_with_count["pr_reid_s4"],
    bins=[0, 0.05, 0.1, 0.2, 0.33, 0.5, float('inf')],
    labels=["<5%", "<10%", "<20%", "<33%", "<50%", ">50%"]
)

# Create table

# List risk groups
risk_groups = ["<5%", "<10%", "<20%", "<33%", "<50%", ">50%"]

# Create dictionary to store data for each s value
summary_data = {"risk": risk_groups}

# Iterate through each s value
for s in range(1, 5):
    percentages = []
    for risk_group in risk_groups:
        column_name = f"risk%_group_s{s}"
        count = (equivalence_classes_with_count[column_name] == risk_group).sum()
        total = len(equivalence_classes_with_count)
        percentage = (count / total) * 100
    # Format values with 2 decimal points and a % sign
    percentages.append(f"{percentage:.2f}%")

    summary_data[f"s{s}"] = percentages

# Create the summary table from the dictionary
summary_table = pd.DataFrame(summary_data)
summary_table.set_index("risk", inplace=True)

# Print table
print(summary_table)

```

	s1	s2	s3	s4
risk				
<5%	84.72%	79.17%	93.06%	79.17%
<10%	11.11%	5.56%	6.94%	5.56%
<20%	4.17%	11.11%	0.00%	11.11%
<33%	0.00%	1.39%	0.00%	1.39%
<50%	0.00%	2.78%	0.00%	2.78%
>50%	0.00%	0.00%	0.00%	0.00%

```
[13]: # === Results - Conclusion Table ===

# Create a dictionary with data
data = {
    's1': ['0.11%', '1.65%', 'pass'],
    's2': ['2.2%', '33.33%', 'fail'],
    's3': ['0.59%', '8.91%', 'pass'],
    's4': ['2.2%', '33.33%', 'fail']
}

# Create a df
df = pd.DataFrame(data, index=['median risk', 'maximum risk', 'assessment'])

# apply color styling
def highlight_cells(val):
    if val == 'pass':
        return 'background-color: green'
    elif val == 'fail':
        return 'background-color: red'
    else:
        return ''

# Apply styling to the df
color_table = df.style.applymap(highlight_cells)

# Display table
color_table
```

```
[13]: <pandas.io.formats.style.Styler at 0x1347f8790>
```

```
[ ]:
```