Data Privacy and Anonymization

```
# show python version
import sys
print(sys.version)

# show python path
print(sys.path)
```

```
3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
['/home/stormbird/Desktop/data-privacy-pres', '/usr/lib/python310.zip', '/usr/lib/python3.10',
'/usr/lib/python3.10/lib-dynload', '', '/home/stormbird/.local/lib/python3.10/site-packages',
'/usr/local/lib/python3.10/dist-packages', '/usr/lib/python3/dist-packages']
```

```
# download and install python packages
```

```
!pip install diffprivlib -q
```

```
!pip install -q numpy pandas matplotlib scikit-learn pandas-profiling phe
```

```
import numpy as np
import pandas as pd
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
from pandas.api.types import CategoricalDtype
from IPython.display import HTML
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from diffprivlib.models import RandomForestClassifier
import diffprivlib as dp
from sklearn.metrics import f1_score, classification_report
from IPython.display import Image
from IPython.core.display import HTML, Image
from IPython.display import IFrame
from pandas_profiling import ProfileReport
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[2], line 9
      7 from sklearn.model_selection import train_test_split
      8 from sklearn.decomposition import PCA
----> 9 from diffprivlib.models import RandomForestClassifier
     10 import diffprivlib as dp
     11 from sklearn.metrics import f1_score, classification_report

ModuleNotFoundError: No module named 'diffprivlib'
```

```
# specify all the datatypes of the dataset
dtypes = {
    "County": object,
    "Gender": CategoricalDtype(),
    "Martial_status": "category",
    "No_of_dependents": np.int32,
    "Age": np.int32,
    "Tier": CategoricalDtype(categories=["Bronze", "Silver", "Gold"],
                             ordered=True),
    "Policy_limit": np.float32,
    "Illness_category": "category",
    "Updated_subscription": object,
    "Updated_subscription": object,
    "Account_withdrawal": object,
    "Retention": np.int32,
    "ID_number": np.int32,
    "Tax_id": object
```

```
}

dtypes2 = {
    "Instrument": object,
    "Reading": object,
    "Observation": np.float32,
    "Status": "category"
}

# load all the datasets and specify the arguments to making parsing easier
insurance_df = pd.read_csv("data/Insurance_data_ke.csv",
                           index_col="id",
                           parse_dates=["Date_of_entry"],
                           dtype=dtypes)
anonymous_df = pd.read_csv("data/Insurance_data_ke.csv", header=None, skiprows=1)
spo2_temp_hr_df = pd.read_csv("data/Organs.csv",
                              parse_dates=["Date"],
                              dtype=dtypes2)
```

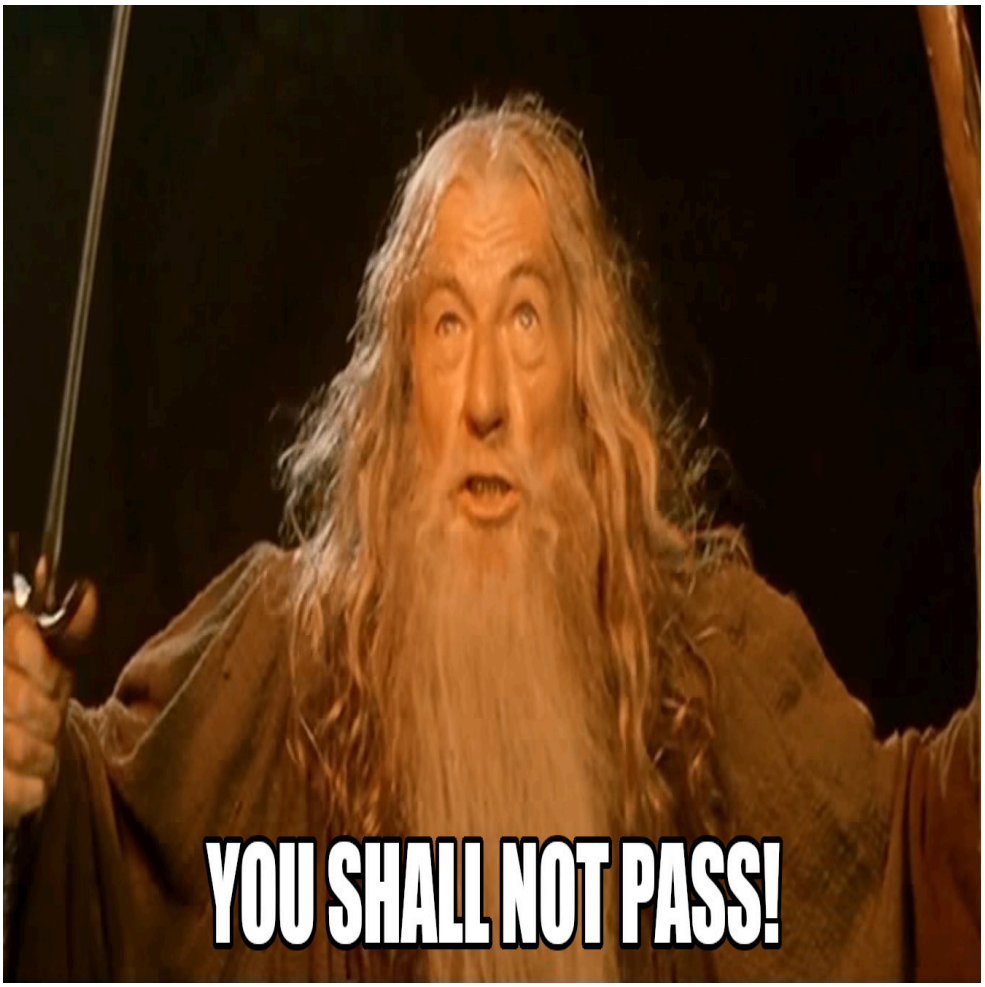
Thank the audience for coming and I will go through it in a breeze. TGIF activities and joining them.

What I'll talk about: * Definitions * Techniques: * Data Deletion! * Masking & Sampling * Privacy Models: K-anonymity, PCA and Differential privacy

Data Privacy

Is the protection of personal data from those who should not have access to it and the ability of individuals to determine who can access their personal information.

Authorized, fair, and legitimate processing of personal information. Credit: *Michelle Dennedy, Jonathan Fox, Tom Finneran, The Privacy Engineer's Manifesto (Apress, 2014), p. 34.*



Personally identifiable information (PIIs)

Sensitive PIIs - info that can be directly linked to a person e.g **biometric data**, **genetic data**, **health status**, **family details**, **National ID number**, **Passport number**, **KRA pin**

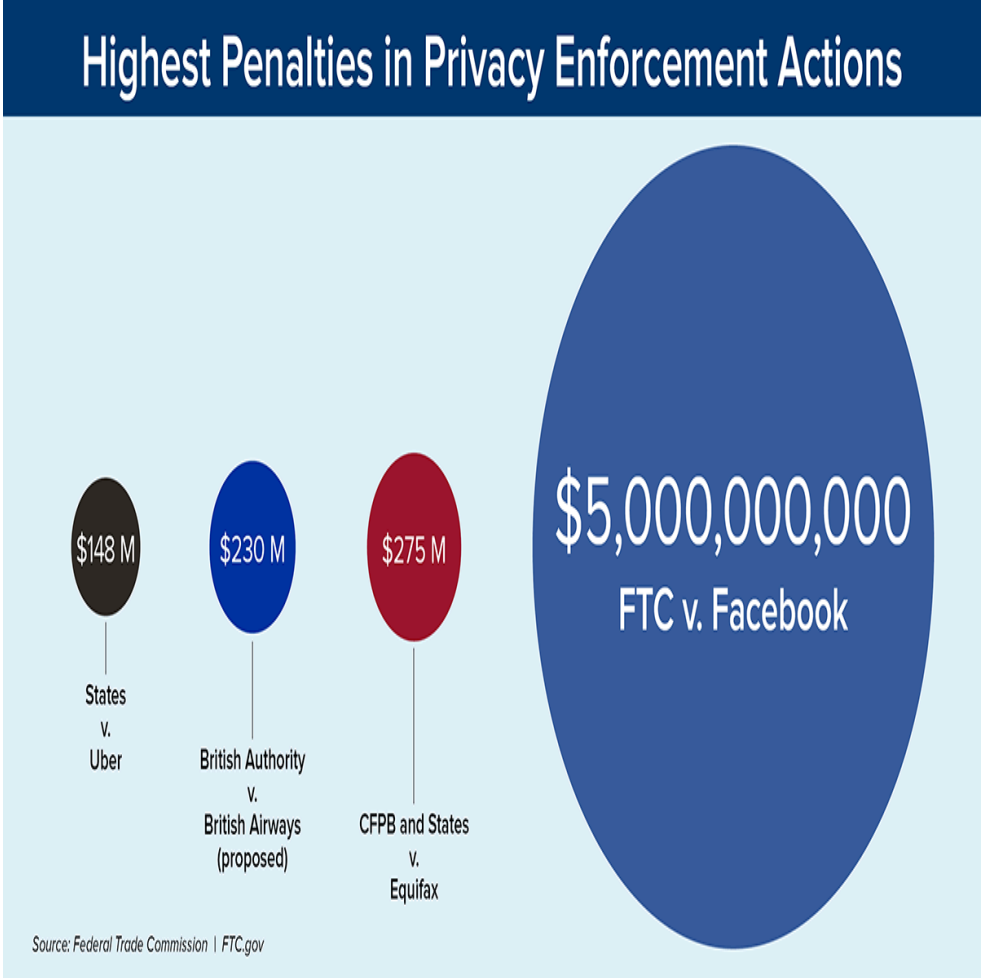
Non-sensitive PIIs- info can't be directly linked to a person as an example **birthday**, **county of origin**

Quasi-identifying - on their own they are not identifying but when combined can be. **Latanya Sweeney** confirmed that combining birth dates, gender and postcodes can be used to identify people in the United States. She also discovered something interesting with her name. Find out. What about Kenya?

Anonymization

Removing PIIs in datasets to keep the individuals Jane/John Does.

Why should we care?



Equifax data breach in 2017, British Airways £20m for data breach affecting more than 400,000 customers, Uber to pay $148 million to settle data breach cover-up with U.S. states and facebook 5 billion dollar penalty

Oppo Kenya was penalized by ODPC for infringing the Data Protection Act, 2019 by publishing unauthorized photographs of data subjects in their promotional materials

Kenya's Office of the Data Privacy Commissioner announced three penalties totaling KES9,375,000 for alleged violations of the Data Protection Act. Each penalty concerns claims of nonconsensual uses of personal data. The largest fine was KES4,550,000 to Roma School for posting pictures of minors without parental consent.

To comply with the Kenya data protection Act and General Data Protection Regulation.

```
IFrame("Kenya Data Protection Act - Quick Guide 2021.pdf", width=350, height=250)
```

Data Deletion

Plan for it:
* Come up with a data inventory and control access
* Follow that up with a segregation strategy: Operational data (10 months) and archival data (5 years) * Create a Retriever(): Gets all the data regarding a specific client. It’s their right. * Create a Destroyer(): Deletes all data regarding a customer from every store e.g database, object storage(object expiration) et cetera * Make a privacy engineering team to handle these.

from Imgflip Meme Generator

Enter the matrix 😎

from Imgflip Meme Generator

Compare Datasets

Ask if the data is real in their eyes?

```
# a summary of the data
anonymous_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001 entries, 0 to 1000
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       1001 non-null   int64 
 1   1       1001 non-null   object
 2   2       1001 non-null   object
 3   3       1001 non-null   object
 4   4       1001 non-null   int64 
 5   5       1001 non-null   int64 
 6   6       1001 non-null   object
 7   7       1001 non-null   int64 
 8   8       1001 non-null   object
 9   9       1001 non-null   object
 10  10      1001 non-null   object
 11  11      1001 non-null   object
 12  12      1001 non-null   int64 
dtypes: int64(5), object(8)
memory usage: 101.8+ KB
```

```
# Masking the data
anonymous_df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	Machakos	Female	Single	2	47	Silver	300000	Chronic	2013-11-14 01:12:01	No	No	14	38074758	N690674664H

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	Kisii	Male	Married	3	39	Silver	300000	Congenital	2015-09-11 02:32:07	Yes	Yes	5	19503833	F384415788F
2	2	Marsabit	Female	Widowed	3	73	Bronze	500000	Chronic	2000-11-24 04:13:56	Yes	No	2	5985350	B871536581P
3	3	Kilifi	Male	Separated	4	52	Gold	300000	Congenital	2006-04-06 12:50:18	No	No	19	23029320	P464662008O
4	4	Marsabit	Female	Married	5	39	Silver	300000	Subacute	2016-12-04 16:49:07	Yes	No	9	22372093	V172966182A

Masking

from Imgflip Meme Generator

```
insurance_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1001 entries, 0 to 1000
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype         
---  ------                --------------  -----         
 0   County                1001 non-null   object        
 1   Gender                1001 non-null   category      
 2   Marital_status        1001 non-null   object        
 3   No_of_dependents      1001 non-null   int32         
 4   Age                   1001 non-null   int32         
 5   Tier                  1001 non-null   category      
 6   Policy_limit          1001 non-null   float32       
 7   Illness_category      1001 non-null   category      
 8   Date_of_entry         1001 non-null   datetime64[ns]
 9   Updated_subscription  1001 non-null   object        
 10  Account_withdrawal    1001 non-null   object        
 11  Retention             1001 non-null   int32         
dtypes: category(3), datetime64[ns](1), float32(1), int32(3), object(4)
memory usage: 65.8+ KB
```

```
# Use pandas profiling for EDA of the dataset
profile = ProfileReport(insurance_df, title="Pandas Profiling Report of the KE insurance company")
profile.to_file("insurance_report.html")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[3], line 2
      1 # Use pandas profiling for EDA of the dataset
----> 2 profile = ProfileReport(insurance_df, title="Pandas Profiling Report of the KE insurance company")
      3 profile.to_file("insurance_report.html")

NameError: name 'ProfileReport' is not defined
```

Switch to HTML report

#HTML(filename='insurance_report.html')

```
# remove columns that are very particular
#insurance_df = insurance_df.drop(["ID_number", "Tax_id"], axis=1)

# see what columns were left
insurance_df.columns
```

```
Index(['County', 'Gender', 'Marital_status', 'No_of_dependents', 'Age', 'Tier',
       'Policy_limit', 'Illness_category', 'Date_of_entry',
       'Updated_subscription', 'Account_withdrawal', 'Retention'],
      dtype='object')
```

Sampling

Taking a subset of the data and calculating a metric on the data for example mean, median, count.

```
# make a subset of the dataframe
insurance_df_sample = insurance_df.sample(n=100)

# finding unique items in the pandas Series and return a probability
insurance_df['Marital_status'].value_counts(normalize=True)
```

```
Separated                0.375624
Single                   0.272727
Married                  0.204795
Widowed                  0.138861
Registered Partnership   0.006993
Divorced                 0.000999
Name: Marital_status, dtype: float64
```

```
mar_status_counts = insurance_df_sample['Marital_status'].value_counts(
    normalize=True)

mar_status_counts
```

```
Separated    0.35
Married      0.28
Single       0.21
Widowed      0.16
Name: Marital_status, dtype: float64

# make a non random sample distribution based on the original dataset
insurance_df_sample["Martial_status"] = np.random.choice(
    mar_status_counts.index,
    p=mar_status_counts.values,
    size=len(insurance_df_sample))

insurance_df_sample['Martial_status'].value_counts(normalize=True)

Separated    0.38
Single       0.22
Married      0.21
Widowed      0.19
Name: Martial_status, dtype: float64
```

K-anonymity

Reducing distinction between groups. K groups share properties.

from Imgflip Meme Generator

```
# arrange dataframe by Age and tier, count instances of both and rename the column
insurance_df.groupby(["Age", "Tier"]).size().reset_index(name="Count")
```

	Age	Tier	Count
0	30	Bronze	7
1	30	Silver	4
2	30	Gold	6

	Age	Tier	Count
3	31	Bronze	9
4	31	Silver	5
...	...	...	...
130	73	Silver	3
131	73	Gold	4
132	74	Bronze	14
133	74	Silver	4
134	74	Gold	4

135 rows × 3 columns

```
# binning data into at most 4 intervals
insurance_df["Age"] = pd.cut(insurance_df["Age"], bins=4)
```

```
# see sample
insurance_df[["Age", "Tier"]]
```

id	Age	Tier
0	(41.0, 52.0]	Silver
1	(29.956, 41.0]	Silver
2	(63.0, 74.0]	Bronze
3	(41.0, 52.0]	Gold
4	(29.956, 41.0]	Silver
...	...	...
996	(63.0, 74.0]	Silver
997	(29.956, 41.0]	Bronze
998	(52.0, 63.0]	Bronze
999	(63.0, 74.0]	Gold
1000	(52.0, 63.0]	Silver

1001 rows × 2 columns

```
# redistributes the information
insurance_df["Age"].value_counts(normalize=True)
```

```
(29.956, 41.0]    0.262737
(52.0, 63.0]      0.258741
(41.0, 52.0]      0.252747
(63.0, 74.0]      0.225774
Name: Age, dtype: float64
```

```
# How many specific groups arise should not be 4
k = 4

count_groups = insurance_df.groupby(["Age",
                                     "Tier"]).size().reset_index(name="Count")

count_groups[count_groups['Count'] < k]
```

	Age	Tier	Count

Differential Privacy

Adding statistical noise to your work. Governed by an epsilon parameter which is the **privacy budget**.

from Imgflip Meme Generator

Would you want someone knowing that you have Diabetes insipidus or what is the staple food in Kenya? Low values give less accurate data whereas high values give the most accurate data. How you can add noise resample your data and decide to remove outliers or just multiple the same outlier with a random number maybe sampled from a normal distribution.

```
# make 10 random numbers
np.random.seed(1)
X = np.random.randint(1, 10, size=10, dtype=np.int32)

# specify budget accountant, adjust value to be 10 and see the difference
acc = dp.BudgetAccountant(epsilon=1.0)

# find the mean
print(X)
print(f"Then mean of your array is",
      dp.tools.mean(X, bounds=(1, 9), accountant=acc),
      ",Using the normal mean function", {np.mean(X)})
print("Budget so far ", acc.remaining())  # how much you have spen
```

```
[6 9 6 1 1 2 8 7 3 5]
Then mean of your array is 4.6072488508630265 ,Using the normal mean function {4.8}
Budget so far  (epsilon=1.1102230246251565e-16, delta=1.0)
```

```
# Add this example if someone says they don't get it
# multiply 2 numbers with a random number sampled from a normal distribution & then do a mean
#X[1] = X[1] * np.random.normal()
#X[9] = X[9] * np.random.normal()

# multiply each number with a number sampled from the normal distribution & then do a mean
# X * np.random.normal() # add noise
# X = np.delete(X, [9, 1], 0) # remove the smallest and largest numbers & then do a mean
# np.mean(X)
```

```
# run it again, should not work
#print(f"Then mean of your array is",
#      {dp.tools.mean(X, bounds=(1, 8), accountant=acc)},
#      "Using the normal mean function", {np.mean(X)})
#print("Budget so far ", acc.remaining())

# load dataset into memory
feat_eng_df = pd.read_csv("data/feature_engineered_insurance.csv")

feat_eng_df.columns

Index(['No_of_dependents', 'Age', 'Policy_limit', 'Retention',
       'Date_of_entryYear', 'Date_of_entryMonth', 'Date_of_entryWeek',
       'Date_of_entryDay', 'Date_of_entryDayofweek', 'Date_of_entryDayofyear',
       'Account_withdrawal=No', 'Account_withdrawal=Yes', 'County=Baringo',
       'County=Bomet', 'County=Bungoma', 'County=Busia',
       'County=Elgeyo-Marakwet', 'County=Embu', 'County=Garissa',
       'County=Homa Bay', 'County=Isiolo', 'County=Kajiado', 'County=Kakamega',
       'County=Kericho', 'County=Kiambu', 'County=Kilifi', 'County=Kirinyaga',
       'County=Kisii', 'County=Kisumu', 'County=Kitui', 'County=Kwale',
       'County=Laikipia', 'County=Lamu', 'County=Machakos', 'County=Makueni',
       'County=Mandera', 'County=Marsabit', 'County=Meru', 'County=Migori',
       'County=Mombasa (County)', 'County=Murang'a', 'County=Nairobi (County)',
       'County=Nakuru', 'County=Nandi', 'County=Narok', 'County=Nyamira',
       'County=Nyandarua', 'County=Nyeri', 'County=Samburu', 'County=Siaya',
       'County=Tana River', 'County=Tharaka-Nithi', 'County=Trans-Nzoia',
       'County=Turkana', 'County=Uasin Gishu', 'County=Vihiga', 'County=Wajir',
       'County=West Pokot', 'Date_of_entryIs_month_end',
       'Date_of_entryIs_month_start', 'Date_of_entryIs_quarter_end',
       'Date_of_entryIs_quarter_start', 'Date_of_entryIs_year_end',
       'Date_of_entryIs_year_start', 'Gender=Female', 'Gender=Male',
       'Illness_category=Acute', 'Illness_category=Chronic',
       'Illness_category=Coegnital', 'Illness_category=Subacute',
       'Marital_status=Divorced', 'Marital_status=Married',
       'Marital_status=Registered Partnership', 'Marital_status=Separated',
       'Marital_status=Single', 'Marital_status=Widowed', 'Tier=Bronze',
       'Tier=Gold', 'Tier=Silver', 'Updated_subscription=No',
       'Updated_subscription=Yes', 'id'],
      dtype='object')

# preparing the matrix and the vector
# Dependent variable/predictors: X and independent variable/Target: If the client is going to stopped
        using the service
feat_eng_df2 = feat_eng_df.drop(["id"], axis=1)
x = feat_eng_df2.to_numpy()
y = np.random.choice([0, 1], p=[0.50, 0.50],
                     size=len(feat_eng_df))
y

array([0, 1, 0, ..., 0, 1, 1])

# Review the sample of array
x.shape

(1001, 81)

y.shape

(1001,)

# model validation strategy: Train test split
features_train, features_validation_test, labels_train, labels_validation_test = train_test_split(
    x, y, test_size=0.4, random_state=100)
features_validation, features_test, labels_validation, labels_test = train_test_split(
    features_validation_test,
    labels_validation_test,
    test_size=0.6,
    random_state=100)
```

```
# Specify, fit, Predict paradigm
clf = RandomForestClassifier(n_jobs=-1, epsilon=0.5, random_state=345, verbose=1)

clf.fit(features_train, labels_train)

preds = clf.predict(features_validation_test)

report = classification_report(labels_validation_test, preds)
print(report)
```

```
/home/bens/anaconda3/envs/data-privacy-env/lib/python3.9/site-packages/diffprivlib/models/forest.py:188:
PrivacyLeakWarning: `feature_domains` parameter hasn't been specified, so falling back to determining
domains from the data.
This may result in additional privacy leakage. To ensure differential privacy with no additional privacy
loss, specify `feature_domains` according to the documentation
  warnings.warn(
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:  5.7min finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.

              precision    recall  f1-score   support

           0       0.52      0.74      0.61       198
           1       0.57      0.33      0.42       203

    accuracy                           0.53       401
   macro avg       0.54      0.54      0.51       401
weighted avg       0.54      0.53      0.51       401

[Parallel(n_jobs=4)]: Done  10 out of  10 | elapsed:    0.7s finished
```

Random Forest is a machine learning algorithm which uses a couple of DecisionTrees on random subsets of the data. Among all the features in the data, picks the one that best divides the labels.

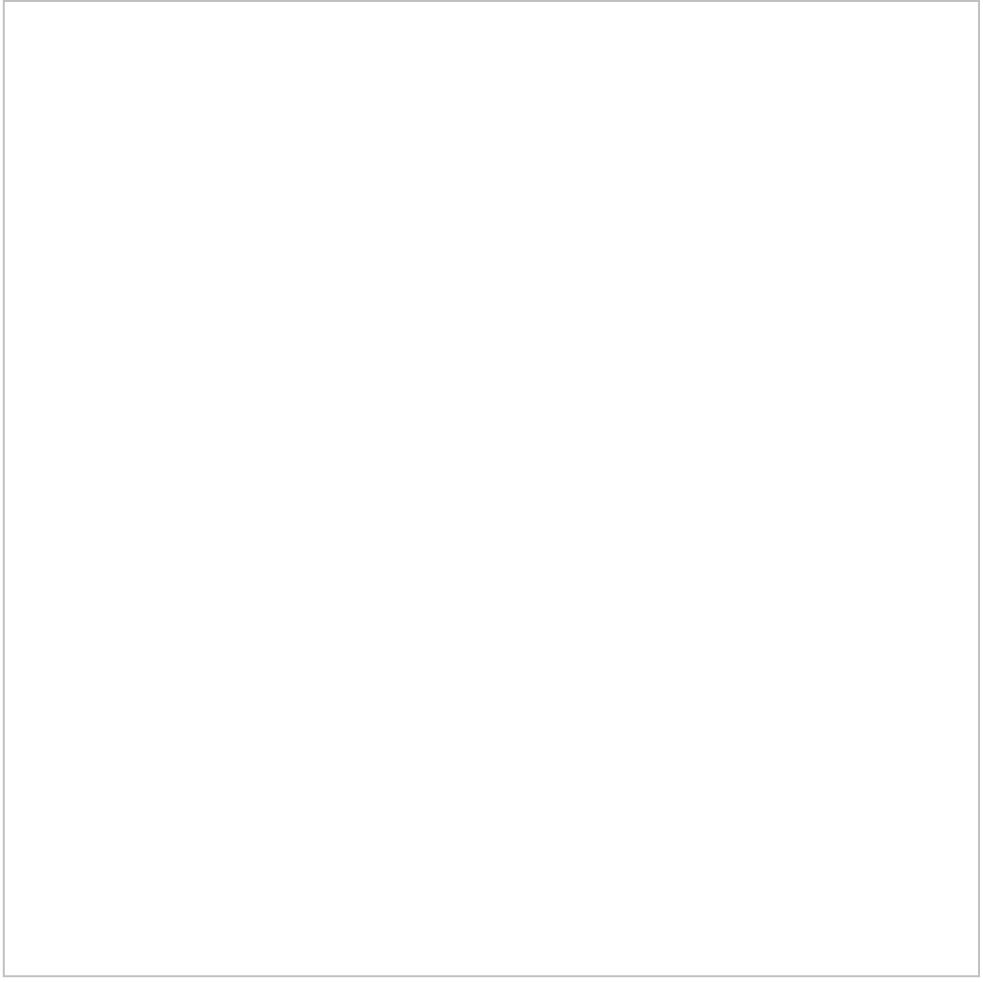
Precision goal is to limit the number of false positives, Recall/Sensitivity how many of the positive points were correctly predicted by the model. Limit the number of false negatives and F1 score combines recall and the precision.

Use the newer version of diffprivlib to tune it better. Try running getting a model without using diffprivlib.

PCA

Principal component Analysis. Very pervasive technique to reduce the dimensions of any dataset and still retaining the properties of the data. You can also use it for data compression.

```
#PATH = "~/Desktop/data_privacy_013/"
Image("Screenshot from 2022-09-10 07-03-38.png",
      width=500,
      height=500)
```



```
def dim_reduction(data):
    pca = PCA(n_components=2)
    return pca.fit_transform(data)

# check how much memory in bytes your columns are using
feat_eng_df2.memory_usage()

Index                        128
No_of_dependents            8008
Age                         8008
Policy_limit                8008
Retention                   8008
                            ...
Tier=Bronze                 8008
Tier=Gold                   8008
Tier=Silver                 8008
Updated_subscription=No     8008
Updated_subscription=Yes    8008
Length: 82, dtype: int64

# apply pca
pca_mat = dim_reduction(feat_eng_df2)

# change the resultant matrix to a dataframe
pca_df = pd.DataFrame(pca_mat)

pca_df.memory_usage()

Index     128
0        8008
1        8008
dtype: int64
```

Moved from 82 columns to 2. Notice: the number of bytes in the column resembles that of the original column entries.

Embeddings

Embeddings are a way to represent data in a lower dimension. They consist of a vector of numbers that represent the data. They are used in Natural Language Processing, Computer Vision and Recommendation Systems. They are commonly used in the transformer architecture. To get a representation of data.

Examples include Word2Vec, GloVe, and FastText. We'll use numpy to create embeddings for our data. Then use T-SNE to visualize the embeddings. It keep things that are close in high dimensions close in 2D or 3D as well.

```
# Define the categorical variables
# NB: The order of the categories is important for the embeddings and the duplicate values are
        intentional
gender = np.array(['Male', 'Female', 'Male', 'Female'])
occupation = np.array(['Engineer', 'Doctor', 'Teacher', 'Engineer'])
education_level = np.array(['Bachelor', 'Doctorate', 'Masters', 'Bachelor'])

# The dimensionality of the embedding vectors
embedding_dim = 3

# Function to generate random embeddings
def generate_embeddings(categories:str, embedding_dim:int):
    """
    Generate random embeddings for categorical variables.

    Parameters
    ----------
    categories : str
        Array of categorical variables.
    embedding_dim : int
        The dimensionality of the embedding vectors.

    Returns
    -------
    dict
        A dictionary where the keys are the unique categories from the input
        and the values are the corresponding embeddings.

    Example
    -------
    >>> categories = np.array(['Male', 'Female'])
    >>> embedding_dim = 3
    >>> embeddings = generate_embeddings(categories, embedding_dim)
    >>> print(embeddings)
    {'Male': array([0.1, 0.2, 0.3]), 'Female': array([0.4, 0.5, 0.6])}
    """
    category_embeddings = {}
    for category in np.unique(categories):
        category_embeddings[category] = np.random.rand(embedding_dim)
    return category_embeddings

# Create embeddings for the categorical variables
gender_embedding = generate_embeddings(gender, embedding_dim)
occupation_embedding = generate_embeddings(occupation, embedding_dim)
education_embedding = generate_embeddings(education_level, embedding_dim)

# Encode the categorical variables using embeddings
# Find the category in the dictionary and use it to get the corresponding embedding
encoded_gender = gender_embedding[gender[0]]
encoded_occupation = occupation_embedding[occupation[0]]
encoded_education_level = education_embedding[education_level[0]]

# Combine all embeddings into a single array for t-SNE
embeddings = np.array(list(gender_embedding.values()) + list(occupation_embedding.values()) +
        list(education_embedding.values()))
```

```
# Print the encoded values
print("Encoded Gender:", encoded_gender)
print("Encoded Occupation:", encoded_occupation)
print("Encoded Education Level:", encoded_education_level)
```

```
Encoded Gender: [0.11183795 0.4517675  0.33624615]
Encoded Occupation: [0.0153379  0.36029145 0.26005155]
Encoded Education Level: [0.53302015 0.86031051 0.83079   ]
```

This is how we can use embeddings to represent data in a numerical form. Besides, without the labels, we don't know what the data is about. You can use this to cluster the data and see if you can get some insights from it. Let's try to cluster the data and see if we can get some insights from it.

```
# Categories for labeling in the plot
categories = list(gender_embedding.keys()) + list(occupation_embedding.keys()) +
        list(education_embedding.keys())
category_labels = ['Gender'] * len(gender_embedding) + ['Occupation'] * len(occupation_embedding) +
        ['Education'] * len(education_embedding)
colors = {'Gender': 'red', 'Occupation': 'blue', 'Education': 'green'}

# Apply t-SNE for dimensionality reduction
# n_components: The number of dimensions to reduce to.
# perplexity: The number of nearest neighbors to consider.
tsne = TSNE(n_components=2, perplexity=3, random_state=42)
embeddings_2d = tsne.fit_transform(embeddings)

# Plot the embeddings in 2D space
plt.figure(figsize=(10, 8))
for i, category in enumerate(categories):
    plt.scatter(embeddings_2d[i, 0], embeddings_2d[i, 1], c=colors[category_labels[i]],
        label=category_labels[i] if i == 0 or category_labels[i] != category_labels[i - 1] else "")

# Annotate the points with labels
for i, txt in enumerate(categories):
    plt.annotate(txt, (embeddings_2d[i, 0], embeddings_2d[i, 1]), fontsize=9)

plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('t-SNE Visualization of Random Embeddings')
plt.legend(loc='best')
plt.show()
```

```
/home/bens/anaconda3/envs/data-privacy-env/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:780:
FutureWarning: The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
/home/bens/anaconda3/envs/data-privacy-env/lib/python3.9/site-packages/sklearn/manifold/_t_sne.py:790:
FutureWarning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  warnings.warn(
```

(Note that this will change depending on your data.)

When *Engineer*, *Doctor*, and *Doctorate* are close to each other in the plot, it tells us that these roles share some key features or characteristics in our data. Maybe they all need high education levels or specialized skills.

On the other hand, if *Teacher* is far from these clustered points, it suggests that being a teacher has different characteristics or features in the data.

Visualization helps us see hidden patterns in our data. For instance, seeing similar roles cluster together can tell us something important about the relationships between these roles, like educational background or job skills.

It helps us understand how our embedding model (which converts categories to numbers) is doing in capturing these patterns.

Federated Learning

Federated learning is a machine learning technique that trains an algorithm across multiple devices or servers holding local data samples, without exchanging them. This approach stands in contrast to traditional centralized machine learning techniques where all data is uploaded to one server.

We'll need to download some data from a repository about spam and ham emails. Note: the next cell applies best practices when it comes to getting data from somewhere. Retries, timeouts, and error handling. These will get you far in your data science journey.

```
!for i in {1..3}; do wget --timeout=300 --tries=3 https://github.com/Shuyib/Grokking-Deep-
        Learning/raw/master/ham.txt -O data/ham.txt && wget --timeout=300 --tries=3
        https://github.com/Shuyib/Grokking-Deep-Learning/raw/master/spam.txt -O data/spam.txt && break
        || sleep 5; done
```

```
--2024-06-20 10:54:33--  https://github.com/Shuyib/Grokking-Deep-Learning/raw/master/ham.txt
Resolving github.com (github.com)... 20.87.245.0
Connecting to github.com (github.com)|20.87.245.0|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/Shuyib/Grokking-Deep-Learning/master/ham.txt [following]
--2024-06-20 10:54:36--  https://raw.githubusercontent.com/Shuyib/Grokking-Deep-Learning/master/ham.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.108.133,
185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 21180948 (20M) [text/plain]
Saving to: ‘data/ham.txt’

data/ham.txt        100%[===================>]  20.20M   241KB/s    in 76s

2024-06-20 10:55:56 (273 KB/s) - ‘data/ham.txt’ saved [21180948/21180948]

--2024-06-20 10:55:56--  https://github.com/Shuyib/Grokking-Deep-Learning/raw/master/spam.txt
Resolving github.com (github.com)... 20.87.245.0
Connecting to github.com (github.com)|20.87.245.0|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/Shuyib/Grokking-Deep-Learning/master/spam.txt [following]
--2024-06-20 10:55:58--  https://raw.githubusercontent.com/Shuyib/Grokking-Deep-Learning/master/spam.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.109.133,
185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10843548 (10M) [application/octet-stream]
Saving to: ‘data/spam.txt’

data/spam.txt       100%[===================>]  10.34M   809KB/s    in 17s

2024-06-20 10:56:18 (615 KB/s) - ‘data/spam.txt’ saved [10843548/10843548]
```

```
# Utility functions for the neural network

class Tensor (object):

    def __init__(self,data,
                 autograd=False,
                 creators=None,
                 creation_op=None,
                 id=None):

        self.data = np.array(data)
        self.autograd = autograd
        self.grad = None

        if(id is None):
            self.id = np.random.randint(0,1000000000)
        else:
            self.id = id

        self.creators = creators
```

```
    self.creation_op = creation_op
    self.children = {}

    if(creators is not None):
        for c in creators:
            if(self.id not in c.children):
                c.children[self.id] = 1
            else:
                c.children[self.id] += 1

def all_children_grads_accounted_for(self):
    for id,cnt in self.children.items():
        if(cnt != 0):
            return False
    return True

def backward(self,grad=None, grad_origin=None):
    if(self.autograd):

        if(grad is None):
            grad = Tensor(np.ones_like(self.data))

        if(grad_origin is not None):
            if(self.children[grad_origin.id] == 0):
                return
                print(self.id)
                print(self.creation_op)
                print(len(self.creators))
                for c in self.creators:
                    print(c.creation_op)
                raise Exception("cannot backprop more than once")
            else:
                self.children[grad_origin.id] -= 1

        if(self.grad is None):
            self.grad = grad
        else:
            self.grad += grad

        # grads must not have grads of their own
        assert grad.autograd == False

        # only continue backpropping if there's something to
        # backprop into and if all gradients (from children)
        # are accounted for override waiting for children if
        # "backprop" was called on this variable directly
        if(self.creators is not None and
           (self.all_children_grads_accounted_for() or
            grad_origin is None)):

            if(self.creation_op == "add"):
                self.creators[0].backward(self.grad, self)
                self.creators[1].backward(self.grad, self)

            if(self.creation_op == "sub"):
                self.creators[0].backward(Tensor(self.grad.data), self)
                self.creators[1].backward(Tensor(self.grad.__neg__().data), self)

            if(self.creation_op == "mul"):
                new = self.grad * self.creators[1]
                self.creators[0].backward(new , self)
                new = self.grad * self.creators[0]
                self.creators[1].backward(new, self)

            if(self.creation_op == "mm"):
                c0 = self.creators[0]
```

```
                c1 = self.creators[1]
                new = self.grad.mm(c1.transpose())
                c0.backward(new)
                new = self.grad.transpose().mm(c0).transpose()
                c1.backward(new)

            if(self.creation_op == "transpose"):
                self.creators[0].backward(self.grad.transpose())

            if("sum" in self.creation_op):
                dim = int(self.creation_op.split("_")[1])
                self.creators[0].backward(self.grad.expand(dim,
                                                           self.creators[0].data.shape[dim]))

            if("expand" in self.creation_op):
                dim = int(self.creation_op.split("_")[1])
                self.creators[0].backward(self.grad.sum(dim))

            if(self.creation_op == "neg"):
                self.creators[0].backward(self.grad.__neg__())

            if(self.creation_op == "sigmoid"):
                ones = Tensor(np.ones_like(self.grad.data))
                self.creators[0].backward(self.grad * (self * (ones - self)))

            if(self.creation_op == "tanh"):
                ones = Tensor(np.ones_like(self.grad.data))
                self.creators[0].backward(self.grad * (ones - (self * self)))

            if(self.creation_op == "index_select"):
                new_grad = np.zeros_like(self.creators[0].data)
                indices_ = self.index_select_indices.data.flatten()
                grad_ = grad.data.reshape(len(indices_), -1)
                for i in range(len(indices_)):
                    new_grad[indices_[i]] += grad_[i]
                self.creators[0].backward(Tensor(new_grad))

            if(self.creation_op == "cross_entropy"):
                dx = self.softmax_output - self.target_dist
                self.creators[0].backward(Tensor(dx))

    def __add__(self, other):
        if(self.autograd and other.autograd):
            return Tensor(self.data + other.data,
                          autograd=True,
                          creators=[self,other],
                          creation_op="add")
        return Tensor(self.data + other.data)

    def __neg__(self):
        if(self.autograd):
            return Tensor(self.data * -1,
                          autograd=True,
                          creators=[self],
                          creation_op="neg")
        return Tensor(self.data * -1)

    def __sub__(self, other):
        if(self.autograd and other.autograd):
            return Tensor(self.data - other.data,
                          autograd=True,
                          creators=[self,other],
                          creation_op="sub")
        return Tensor(self.data - other.data)

    def __mul__(self, other):
```

```
    if(self.autograd and other.autograd):
        return Tensor(self.data * other.data,
                      autograd=True,
                      creators=[self,other],
                      creation_op="mul")
    return Tensor(self.data * other.data)

def sum(self, dim):
    if(self.autograd):
        return Tensor(self.data.sum(dim),
                      autograd=True,
                      creators=[self],
                      creation_op="sum_"+str(dim))
    return Tensor(self.data.sum(dim))

def expand(self, dim,copies):

    trans_cmd = list(range(0,len(self.data.shape)))
    trans_cmd.insert(dim,len(self.data.shape))
    new_data = self.data.repeat(copies).reshape(list(self.data.shape) +
    [copies]).transpose(trans_cmd)

    if(self.autograd):
        return Tensor(new_data,
                      autograd=True,
                      creators=[self],
                      creation_op="expand_"+str(dim))
    return Tensor(new_data)

def transpose(self):
    if(self.autograd):
        return Tensor(self.data.transpose(),
                      autograd=True,
                      creators=[self],
                      creation_op="transpose")

    return Tensor(self.data.transpose())

def mm(self, x):
    if(self.autograd):
        return Tensor(self.data.dot(x.data),
                      autograd=True,
                      creators=[self,x],
                      creation_op="mm")
    return Tensor(self.data.dot(x.data))

def sigmoid(self):
    if(self.autograd):
        return Tensor(1 / (1 + np.exp(-self.data)),
                      autograd=True,
                      creators=[self],
                      creation_op="sigmoid")
    return Tensor(1 / (1 + np.exp(-self.data)))

def tanh(self):
    if(self.autograd):
        return Tensor(np.tanh(self.data),
                      autograd=True,
                      creators=[self],
                      creation_op="tanh")
    return Tensor(np.tanh(self.data))

def index_select(self, indices):

    if(self.autograd):
        new = Tensor(self.data[indices.data],
```

```
                        autograd=True,
                        creators=[self],
                        creation_op="index_select")
            new.index_select_indices = indices
            return new
        return Tensor(self.data[indices.data])

    def softmax(self):
        temp = np.exp(self.data)
        softmax_output = temp / np.sum(temp,
                                       axis=len(self.data.shape)-1,
                                       keepdims=True)
        return softmax_output

    def cross_entropy(self, target_indices):

        temp = np.exp(self.data)
        softmax_output = temp / np.sum(temp,
                                       axis=len(self.data.shape)-1,
                                       keepdims=True)

        t = target_indices.data.flatten()
        p = softmax_output.reshape(len(t),-1)
        target_dist = np.eye(p.shape[1])[t]
        loss = -(np.log(p) * (target_dist)).sum(1).mean()

        if(self.autograd):
            out = Tensor(loss,
                         autograd=True,
                         creators=[self],
                         creation_op="cross_entropy")
            out.softmax_output = softmax_output
            out.target_dist = target_dist
            return out

        return Tensor(loss)


    def __repr__(self):
        return str(self.data.__repr__())

    def __str__(self):
        return str(self.data.__str__())

class Layer(object):

    def __init__(self):
        self.parameters = list()

    def get_parameters(self):
        return self.parameters


class SGD(object):

    def __init__(self, parameters, alpha=0.1):
        self.parameters = parameters
        self.alpha = alpha

    def zero(self):
        for p in self.parameters:
            p.grad.data *= 0

    def step(self, zero=True):

        for p in self.parameters:
```

```
            p.data -= p.grad.data * self.alpha

            if(zero):
                p.grad.data *= 0


class Linear(Layer):

    def __init__(self, n_inputs, n_outputs, bias=True):
        super().__init__()

        self.use_bias = bias

        W = np.random.randn(n_inputs, n_outputs) * np.sqrt(2.0/(n_inputs))
        self.weight = Tensor(W, autograd=True)
        if(self.use_bias):
            self.bias = Tensor(np.zeros(n_outputs), autograd=True)

        self.parameters.append(self.weight)

        if(self.use_bias):
            self.parameters.append(self.bias)

    def forward(self, input):
        if(self.use_bias):
            return input.mm(self.weight)+self.bias.expand(0,len(input.data))
        return input.mm(self.weight)


class Sequential(Layer):

    def __init__(self, layers=list()):
        super().__init__()

        self.layers = layers

    def add(self, layer):
        self.layers.append(layer)

    def forward(self, input):
        for layer in self.layers:
            input = layer.forward(input)
        return input

    def get_parameters(self):
        params = list()
        for l in self.layers:
            params += l.get_parameters()
        return params


class Embedding(Layer):

    def __init__(self, vocab_size, dim):
        super().__init__()

        self.vocab_size = vocab_size
        self.dim = dim

        # this random initialiation style is just a convention from word2vec
        self.weight = Tensor((np.random.rand(vocab_size, dim) - 0.5) / dim, autograd=True)

        self.parameters.append(self.weight)

    def forward(self, input):
        return self.weight.index_select(input)
```

```
class Tanh(Layer):
    def __init__(self):
        super().__init__()

    def forward(self, input):
        return input.tanh()


class Sigmoid(Layer):
    def __init__(self):
        super().__init__()

    def forward(self, input):
        return input.sigmoid()


class CrossEntropyLoss(object):

    def __init__(self):
        super().__init__()

    def forward(self, input, target):
        return input.cross_entropy(target)

class MSELoss(object):

    def __init__(self):
        super().__init__()

    def forward(self, input, target):
        dif = input - target
        return (dif * dif).sum(0)

class RNNCell(Layer):

    def __init__(self, n_inputs, n_hidden, n_output, activation='sigmoid'):
        super().__init__()

        self.n_inputs = n_inputs
        self.n_hidden = n_hidden
        self.n_output = n_output

        if(activation == 'sigmoid'):
            self.activation = Sigmoid()
        elif(activation == 'tanh'):
            self.activation == Tanh()
        else:
            raise Exception("Non-linearity not found")

        self.w_ih = Linear(n_inputs, n_hidden)
        self.w_hh = Linear(n_hidden, n_hidden)
        self.w_ho = Linear(n_hidden, n_output)

        self.parameters += self.w_ih.get_parameters()
        self.parameters += self.w_hh.get_parameters()
        self.parameters += self.w_ho.get_parameters()

    def forward(self, input, hidden):
        from_prev_hidden = self.w_hh.forward(hidden)
        combined = self.w_ih.forward(input) + from_prev_hidden
        new_hidden = self.activation.forward(combined)
        output = self.w_ho.forward(new_hidden)
        return output, new_hidden

    def init_hidden(self, batch_size=1):
```

```
        return Tensor(np.zeros((batch_size,self.n_hidden)), autograd=True)

class LSTMCell(Layer):

    def __init__(self, n_inputs, n_hidden, n_output):
        super().__init__()

        self.n_inputs = n_inputs
        self.n_hidden = n_hidden
        self.n_output = n_output

        self.xf = Linear(n_inputs, n_hidden)
        self.xi = Linear(n_inputs, n_hidden)
        self.xo = Linear(n_inputs, n_hidden)
        self.xc = Linear(n_inputs, n_hidden)

        self.hf = Linear(n_hidden, n_hidden, bias=False)
        self.hi = Linear(n_hidden, n_hidden, bias=False)
        self.ho = Linear(n_hidden, n_hidden, bias=False)
        self.hc = Linear(n_hidden, n_hidden, bias=False)

        self.w_ho = Linear(n_hidden, n_output, bias=False)

        self.parameters += self.xf.get_parameters()
        self.parameters += self.xi.get_parameters()
        self.parameters += self.xo.get_parameters()
        self.parameters += self.xc.get_parameters()

        self.parameters += self.hf.get_parameters()
        self.parameters += self.hi.get_parameters()
        self.parameters += self.ho.get_parameters()
        self.parameters += self.hc.get_parameters()

        self.parameters += self.w_ho.get_parameters()

    def forward(self, input, hidden):

        prev_hidden = hidden[0]
        prev_cell = hidden[1]

        f = (self.xf.forward(input) + self.hf.forward(prev_hidden)).sigmoid()
        i = (self.xi.forward(input) + self.hi.forward(prev_hidden)).sigmoid()
        o = (self.xo.forward(input) + self.ho.forward(prev_hidden)).sigmoid()
        g = (self.xc.forward(input) + self.hc.forward(prev_hidden)).tanh()
        c = (f * prev_cell) + (i * g)

        h = o * c.tanh()

        output = self.w_ho.forward(h)
        return output, (h, c)

    def init_hidden(self, batch_size=1):
        init_hidden = Tensor(np.zeros((batch_size,self.n_hidden)), autograd=True)
        init_cell = Tensor(np.zeros((batch_size,self.n_hidden)), autograd=True)
        init_hidden.data[:,0] += 1
        init_cell.data[:,0] += 1
        return (init_hidden, init_cell)

# preparation of the data
# From the book Grokking Deep Learning by Andrew W. Trask, Manning Publications
from collections import Counter
import random
import codecs


with codecs.open('data/spam.txt', "r",encoding='utf-8', errors='ignore') as fdata:
    raw = fdata.readlines()
```

```
vocab = set()

spam = list()
for row in raw:
    spam.append(set(row[:-2].split(" ")))
    for word in spam[-1]:
        vocab.add(word)

import codecs
with codecs.open('data/ham.txt', "r",encoding='utf-8', errors='ignore') as fdata:
    raw = fdata.readlines()

ham = list()
for row in raw:
    ham.append(set(row[:-2].split(" ")))
    for word in ham[-1]:
        vocab.add(word)

vocab.add("<unk>")

vocab = list(vocab)
w2i = {}
for i,w in enumerate(vocab):
    w2i[w] = i

def to_indices(input, l=500):
    indices = list()
    for line in input:
        if(len(line) < l):
            line = list(line) + ["<unk>"] * (l - len(line))
            idxs = list()
            for word in line:
                idxs.append(w2i[word])
            indices.append(idxs)
    return indices

spam_idx = to_indices(spam)
ham_idx = to_indices(ham)

train_spam_idx = spam_idx[0:-1000]
train_ham_idx = ham_idx[0:-1000]

test_spam_idx = spam_idx[-1000:]
test_ham_idx = ham_idx[-1000:]

train_data = list()
train_target = list()

test_data = list()
test_target = list()

for i in range(max(len(train_spam_idx),len(train_ham_idx))):
    train_data.append(train_spam_idx[i%len(train_spam_idx)])
    train_target.append([1])

    train_data.append(train_ham_idx[i%len(train_ham_idx)])
    train_target.append([0])

for i in range(max(len(test_spam_idx),len(test_ham_idx))):
    test_data.append(test_spam_idx[i%len(test_spam_idx)])
    test_target.append([1])

    test_data.append(test_ham_idx[i%len(test_ham_idx)])
    test_target.append([0])
```

```
def train(model, input_data, target_data, batch_size=500, iterations=5):

    criterion = MSELoss()
    optim = SGD(parameters=model.get_parameters(), alpha=0.01)
    bs = batch_size
    n_batches = int(len(input_data) / batch_size)
    for iter in range(iterations):
        iter_loss = 0
        for b_i in range(n_batches):

            # padding token should stay at 0
            model.weight.data[w2i['<unk>']] *= 0
            input = Tensor(input_data[b_i*bs:(b_i+1)*bs], autograd=True)
            target = Tensor(target_data[b_i*bs:(b_i+1)*bs], autograd=True)

            pred = model.forward(input).sum(1).sigmoid()
            loss = criterion.forward(pred,target)
            loss.backward()
            optim.step()

            iter_loss += loss.data[0] / bs

            sys.stdout.write("\r\tLoss:" + str(iter_loss / (b_i+1)))
        print()
    return model

def test(model, test_input, test_output):

    model.weight.data[w2i['<unk>']] *= 0

    input = Tensor(test_input, autograd=True)
    target = Tensor(test_output, autograd=True)

    pred = model.forward(input).sum(1).sigmoid()
    return ((pred.data > 0.5) == target.data).mean()

model = Embedding(vocab_size=len(vocab), dim=1)
model.weight.data *= 0
criterion = MSELoss()
optim = SGD(parameters=model.get_parameters(), alpha=0.01)

for i in range(3):
    model = train(model, train_data, train_target, iterations=1)
    print("% Correct on Test Set: " + str(test(model, test_data, test_target)*100))

    Loss:0.037140416860871466
% Correct on Test Set: 98.65
    Loss:0.011258669226059108
% Correct on Test Set: 99.15
    Loss:0.008068268387986223
% Correct on Test Set: 99.45

# Basic Federated Learning
import copy
bob = (train_data[0:1000], train_target[0:1000])
alice = (train_data[1000:2000], train_target[1000:2000])
sue = (train_data[2000:], train_target[2000:])

model = Embedding(vocab_size=len(vocab), dim=1)
model.weight.data *= 0

for i in range(3):
    print("Starting Training Round...")
    print("\tStep 1: send the model to Bob")
    bob_model = train(copy.deepcopy(model), bob[0], bob[1], iterations=1)

    print("\n\tStep 2: send the model to Alice")
```

```
    alice_model = train(copy.deepcopy(model), alice[0], alice[1], iterations=1)

    print("\n\tStep 3: Send the model to Sue")
    sue_model = train(copy.deepcopy(model), sue[0], sue[1], iterations=1)

    print("\n\tAverage Everyone's New Models")
    model.weight.data = (bob_model.weight.data + \
                         alice_model.weight.data + \
                         sue_model.weight.data)/3

    print("\t% Correct on Test Set: " + \
          str(test(model, test_data, test_target)*100))

    print("\nRepeat!!\n")
```

```
Starting Training Round...
    Step 1: send the model to Bob
    Loss:0.21908166249699718

    Step 2: send the model to Alice
    Loss:0.2937106899184867

    Step 3: Send the model to Sue
    Loss:0.033339966977175894

    Average Everyone's New Models
    % Correct on Test Set: 84.05

Repeat!!

Starting Training Round...
    Step 1: send the model to Bob
    Loss:0.06625367483630413

    Step 2: send the model to Alice
    Loss:0.09595374225556821

    Step 3: Send the model to Sue
    Loss:0.020290247881140743

    Average Everyone's New Models
    % Correct on Test Set: 92.25

Repeat!!

Starting Training Round...
    Step 1: send the model to Bob
    Loss:0.030819682914453833

    Step 2: send the model to Alice
    Loss:0.03580324891736099

    Step 3: Send the model to Sue
    Loss:0.015368461608470256

    Average Everyone's New Models
    % Correct on Test Set: 98.8

Repeat!!
```

Federated learning with Homomorphic Encryption

```
import phe

public_key, private_key = phe.generate_paillier_keypair(n_length=1024)

# encrypt the number "5"
x = public_key.encrypt(5)
```

```
# encrypt the number "3"
y = public_key.encrypt(3)

# add the two encrypted values
z = x + y

# decrypt the result
z_ = private_key.decrypt(z)
print("The Answer: " + str(z_))
```

```
The Answer: 8
```

```
model = Embedding(vocab_size=len(vocab), dim=1)
model.weight.data *= 0

# note that in production the n_length should be at least 1024
public_key, private_key = phe.generate_paillier_keypair(n_length=128)

def train_and_encrypt(model, input, target, pubkey):
    new_model = train(copy.deepcopy(model), input, target, iterations=1)

    encrypted_weights = list()
    for val in new_model.weight.data[:,0]:
        encrypted_weights.append(public_key.encrypt(val))
    ew = np.array(encrypted_weights).reshape(new_model.weight.data.shape)

    return ew

for i in range(3):
    print("\nStarting Training Round...")
    print("\tStep 1: send the model to Bob")
    bob_encrypted_model = train_and_encrypt(copy.deepcopy(model),
                                            bob[0], bob[1], public_key)

    print("\n\tStep 2: send the model to Alice")
    alice_encrypted_model = train_and_encrypt(copy.deepcopy(model),
                                              alice[0], alice[1], public_key)

    print("\n\tStep 3: Send the model to Sue")
    sue_encrypted_model = train_and_encrypt(copy.deepcopy(model),
                                            sue[0], sue[1], public_key)

    print("\n\tStep 4: Bob, Alice, and Sue send their")
    print("\tencrypted models to each other.")
    aggregated_model = bob_encrypted_model + \
                       alice_encrypted_model + \
                       sue_encrypted_model

    print("\n\tStep 5: only the aggregated model")
    print("\tis sent back to the model owner who")
    print("\t can decrypt it.")
    raw_values = list()
    for val in sue_encrypted_model.flatten():
        raw_values.append(private_key.decrypt(val))
    model.weight.data = np.array(raw_values).reshape(model.weight.data.shape)/3

    print("\t% Correct on Test Set: " + \
              str(test(model, test_data, test_target)*100))
```

```
Starting Training Round...
    Step 1: send the model to Bob
    Loss:0.21908166249699718

    Step 2: send the model to Alice
    Loss:0.2937106899184867

    Step 3: Send the model to Sue
```

```
    Loss:0.033339966977175894

    Step 4: Bob, Alice, and Sue send their
    encrypted models to each other.

    Step 5: only the aggregated model
    is sent back to the model owner who
     can decrypt it.
    % Correct on Test Set: 98.75

Starting Training Round...
    Step 1: send the model to Bob
    Loss:0.06366414053035604

    Step 2: send the model to Alice
    Loss:0.061000357913513055

    Step 3: Send the model to Sue
    Loss:0.025483920416627266

    Step 4: Bob, Alice, and Sue send their
    encrypted models to each other.

    Step 5: only the aggregated model
    is sent back to the model owner who
     can decrypt it.
    % Correct on Test Set: 99.05000000000001

Starting Training Round...
    Step 1: send the model to Bob
    Loss:0.058477976535441636

    Step 2: send the model to Alice
    Loss:0.05987976552444443

    Step 3: Send the model to Sue
    Loss:0.024763428511034752

    Step 4: Bob, Alice, and Sue send their
    encrypted models to each other.

    Step 5: only the aggregated model
    is sent back to the model owner who
     can decrypt it.
    % Correct on Test Set: 99.15

# How would you scan this dataset with your current knowledge
spo2_temp_hr_df.info()
print("#" * 100)
spo2_temp_hr_df.head()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 693 entries, 0 to 692
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         693 non-null    object
 1   Instrument   693 non-null    object
 2   Reading      693 non-null    object
 3   Observation  693 non-null    float32
 4   Status       693 non-null    category
dtypes: category(1), float32(1), object(3)
memory usage: 20.0+ KB
####################################################################################################
```

	Date	Instrument	Reading	Observation	Status
0	2021/04/04 12:05:00	Pulse oximeter	HR	62.000000	Well
1	2021/04/04 12:06:00	Pulse oximeter	SpO2	96.000000	Well
2	2021/04/04 12:13:42	Thermometer	Temperature	36.700001	Well

	Date	Instrument	Reading	Observation	Status
3	2021/10/04 09:33:00	Thermometer	Temperature	37.000000	Well
4	2021/10/04 09:33:00	Pulse oximeter	HR	63.000000	Well

```
spo2_temp_hr_df.Status.value_counts(normalize=True)
```

```
Well       0.917749
Tired      0.056277
Allergy    0.012987
anxious    0.008658
Off        0.004329
Name: Status, dtype: float64
```

Summary:

Definitions:
* **Data Privacy**: segregating data and controlling access. “Gatekeeping data”
* **Anonymization**: removing PIIs
* PII information that can identify you either directly sensitive PIIs or indirect Quasi-identifiers. Non-sensitive PIIs are not dangerous but they are still personal.
* **Techniques**:
* **Data Deletion -** is a way of reducing load on your object storage and database. Make a solid plan and talk to legal team to advice. A retriever and a destroyer is a good start. You even save money.

```
* **Masking & Sampling** - storing data without column names in for instance object storage and giving a small subset still retaining properties of the original dataset.

* **K-anonymity** - making groups that exist have similar characteristics reducing reidentification strategies.  Look into l-diversity.

* **Differential privacy ML** - adding noise to your models governed by BudgetAccountant and epsilon parameter.

* **PCA** - reducing dimensions of your data and still retaining the properties of the data.

* **Embeddings** - representing data in a lower dimension.

* **Federated Learning** - training an algorithm across multiple devices or servers holding local data samples, without exchanging them.

* **Federated Learning with Homomorphic Encryption** - training an algorithm across multiple devices or servers holding local data samples, without exchanging them and using homomorphic encryption to secure the data.
```

Learn more

- https://www.manning.com/books/data-privacy
- https://ethics.fast.ai/
- https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf
- https://www.manning.com/books/privacy-preserving-machine-learning
- https://www.manning.com/books/grokking-deep-learning
- https://www.manning.com/books/build-a-career-in-data-science
- https://www.datacamp.com/