profile.csv

Either all 3 attributes (income, gender, age) are present, or all 3 are absent together. For now I drop these rows

```
print("Times all 3 attributes are absent together:",
((profile["income"].isna() == profile["gender"].isna()) ==
(profile["age"].isna() == profile["gender"].isna())).mean())
print("Ratio of records where these 3 attributes are empty: ",
profile["income"].isna().sum()/len(profile))
```

```
Times all 3 attributes are absent together: 1.0
Ratio of records where these 3 attributes are empty: 0.12794117647058822
```

I can add these back in using the approach i mentioned in report1.pdf
Different users have different date of joining and incomes, and incomes usually increase every year. Need to calculate income of users on the latest date in the dataset. I assume a growth rate of 8% PA. Also creating a new feature, days_diff, showing how long a user has been in the dataset

transcript.csv

There are some entries where the dict in the value column has multiple items. Each key gets its own column for better clarity. A single user has multiple entries

Aggregation

I'll keep duplicates for now, can't be sure if these are added accidentally or are valid. Could be an edge case where a user has been accidentally given the same offer/value twice, probably needs more analysis. Do we want this to happen, or do we want this to happen more frequently?

I left merge this with portfolio on offer_ids, which i can group by person id and sort by time to get a chronological history of transactions by each user (let's call this `df`)

After getting dummies for all relevant columns, these are the final attributes for each user:

```
Index(['time', 'amount', 'reward_x', 'age', 'days_diff', 'max_date_income',
       'reward_y', 'difficulty', 'duration', 'event_offer completed',
       'event_offer received', 'event_offer viewed', 'event_transaction',
       'value_amount', 'value_offer_id', 'value_reward',
       'offer_id_0b1e1539f2cc45b7b9fa7c272da2e1d7',
       'offer_id_2298d6c36e964ae4a3e7e9706d1fb8c2',
       'offer_id_2906b810c7d4411798c6938adc9daaa5',
       'offer_id_3f207df678b143eea3cee63160fa8bed',
       'offer_id_4d5c57ea9a6940dd891ad53e9dbe8da0',
```

```
       'offer_id_5a8bc65990b245e5a138643cd4eb9837',
       'offer_id_9b98b8c7a33c4b65b9aebfe6a799e6d9',
       'offer_id_ae264e3637204a6fb9bb56bc8210ddfd',
       'offer_id_f19421c1d4aa40978ebb69ca19b0e20d',
       'offer_id_fafdcd668e3743c1bb461111dcafc2a4', 'gender_F', 'gender_M',
       'gender_O', 'channels_['email', 'mobile', 'social']',
       'channels_['web', 'email', 'mobile', 'social']',
       'channels_['web', 'email', 'mobile']', 'channels_['web', 'email']',
       'offer_type_bogo', 'offer_type_discount', 'offer_type_informational'],
      dtype='object')
```

Minimum entries for any user: 2 Maximum entries for any user: 56

Model

This is now a sort of time forecasting problem
I have any recurrent base model, LSTM, stacked LSTM, etc with all attributes of `df` as input

Architecture 1

For each user, I iterate over their `df`. As soon as I see event_transaction set to 1, [`offer_id_…` columns, 0] from that timestep is added to training labels, while all timesteps before become the corresponding training samples. If event_transaction is set to 0, with probability p, I add all previous timesteps as input, with [`offer_id_…` columns, 1] added to the training labels.
An extra value is appended to each label to consider cases where we don't want to give the user an offer now.
Left padding is applied to the training samples.
I have a Dense layer from my base model that has softmax to predict which predicts the offer_id (or no offer)
CE loss and accuracy are used as metrics to determine performance.

Architecture 2

My input window size here can be a hyperparameter, let's consider it to be 8 with left padding

From this base model, I have 2 parallel Dense layers

Layer1: gives softmax output of length 4 (equivalent to predicting [`'event_offer completed', 'event_offer received', 'event_offer viewed', 'event_transaction'`])
Layer2: gives softmax output over all `offer_id_…`

Custom loss function:

- Calculate cross entropy of layer1 with output1 -> loss1
- Add this loss1 to final loss
- Calculate cross entropy loss of layer2 -> loss 2,
- if layer1 is correct and output1 is equal to [0,0,0,1] (event_transaction)
  - Add loss2 to final loss
- Else
  - Add zeroes_like(loss2) to final loss

Both loss values can be weighted: final loss = x*loss1 + (1-x)*loss2