Data Load & Overview

In [2]:
```
import os
import numpy as np
import pandas as pd
import re
from tqdm import tqdm
import json
import gc

from functools import reduce
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
```

In [2]:
```
df_train = pd.read_csv('./data/Train_rev1.csv')
df_train.head()
```

Out[2]:

	Id	Title	FullDescription	LocationRaw	LocationNormalized	ContractType	ContractTime	Company
0	12612628	Engineering Systems Analyst	Engineering Systems Analyst Dorking Surrey Sal...	Dorking, Surrey, Surrey	Dorking	NaN	permanent	Gregory Martin International
1	12612830	Stress Engineer Glasgow	Stress Engineer Glasgow Salary **** to **** We...	Glasgow, Scotland, Scotland	Glasgow	NaN	permanent	Gregory Martin International
2	12612844	Modelling and simulation analyst	Mathematical Modeller / Simulation Analyst / O...	Hampshire, South East, South East	Hampshire	NaN	permanent	Gregory Martin International
3	12613049	Engineering Systems Analyst / Mathematical Mod...	Engineering Systems Analyst / Mathematical Mod...	Surrey, South East, South East	Surrey	NaN	permanent	Gregory Martin International
4	12613647	Pioneer, Miser Engineering Systems Analyst	Pioneer, Miser Engineering Systems Analyst Do...	Surrey, South East, South East	Surrey	NaN	permanent	Gregory Martin International

In [3]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244768 entries, 0 to 244767
Data columns (total 12 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   Id                  244768 non-null  int64
 1   Title               244767 non-null  object
 2   FullDescription     244768 non-null  object
 3   LocationRaw         244768 non-null  object
 4   LocationNormalized  244768 non-null  object
 5   ContractType        65442 non-null   object
 6   ContractTime        180863 non-null  object
 7   Company             212338 non-null  object
 8   Category            244768 non-null  object
 9   SalaryRaw           244768 non-null  object
 10  SalaryNormalized    244768 non-null  int64
 11  SourceName          244767 non-null  object
dtypes: int64(2), object(10)
memory usage: 22.4+ MB
```

- **Most of the data type are not numeric which requires us for further processing:**
  - These "objects" represent texts (information) about the job advertisement.

  - for columns like **'Category', 'ContractType', 'ContractTime'** we can use simple encoder like one-hot encoder.
  - for columns like **'Title', 'FullDescription'** we might need some NLP techniques' help for better embedding them and get better representing of these feature.
- **Our target:**
  - **SalaryNormalized:** is a numerical feature which will remain the same.

EDA

In [4]:
```
#check missing data
missing_data = df_train.isnull().sum()
missing_data = pd.DataFrame(missing_data, columns=['MissingNum'])
missing_data['Percentage'] = missing_data/len(df_train)*100
missing_data
```

Out[4]:

	MissingNum	Percentage
Id	0	0.000000
Title	1	0.000409
FullDescription	0	0.000000
LocationRaw	0	0.000000
LocationNormalized	0	0.000000
ContractType	179326	73.263662
ContractTime	63905	26.108397
Company	32430	13.249281
Category	0	0.000000
SalaryRaw	0	0.000000
SalaryNormalized	0	0.000000
SourceName	1	0.000409

- First issue pointed out is the **'Huge' missing of 'ContractType'** which we might consider drop this feature.

- While for the features **ContractTime**, and **Company** we might consider adding a third category as 'unknown' to impute.

In [5]:
```
#check locations
location_data = Counter(df_train.LocationNormalized)
location_data_top10 = location_data.most_common(10)
location_data_top10 = pd.DataFrame(location_data_top10, columns=['Location','Count'])
location_data_top10['Percentage'] = location_data_top10['Count']/len(df_train)*100
location_data_top10
```

Out[5]:

	Location	Count	Percentage
0	UK	41093	16.788551
1	London	30522	12.469767
2	South East London	11713	4.785348
3	The City	6678	2.728298
4	Manchester	3516	1.436462
5	Leeds	3401	1.389479
6	Birmingham	3061	1.250572
7	Central London	2607	1.065090
8	West Midlands	2540	1.037717
9	Surrey	2397	0.979295

- Even though LocationNormalized don't have missing data, **But**,
  - What is 'The City'? Why 'UK'(united) compare with 'West Midlands'(area), 'Surrey'(county), 'Leeds'(city).
  - For better utilize the location information we might need to pre-process and clean it by ourselves.
  - Adzuna's normalised location from their own location tree, and they claimed This normaliser is not perfect

In [6]:
```
#check contract
contract_data = Counter(df_train.ContractTime).most_common()
contract_data = pd.DataFrame(contract_data, columns=['Contract','Count'])
contract_data['Percentage'] = contract_data['Count']/len(df_train)*100
contract_data
```

Out [6]:

	Contract	Count	Percentage
0	permanent	151521	61.903925
1	NaN	63905	26.108397
2	contract	29342	11.987678

- We can simply just sign 'unknown' as a third type to NaN for fixing this missing issue
- Even adding 'unknown' there will only have 3 types, which using one-hot to encode is also fine (Sparse matrix concern).

In [7]:
```
# after EDA, we release the memory of the data
del missing_data
del location_data
del contract_data
gc.collect()
```

Out [7]: 0

Data Pre-Processing

Location Process

In [611…
```
import sys
import urllib
import simplejson as json

#API keys
DOMAIN = 'http://api.geonames.org/'
USERNAME = 'soulofshadow'
VALID_KWARGS = ('q', 'name', 'name_equals', 'name_startsWith', 'maxRows', 'startRow',
                'country', 'countryBias', 'continentCode',
                'adminCode1', 'adminCode2', 'adminCode3', 'featureClass', 'featureCode',
                'lang', 'type', 'style', 'isNameRequired', 'tag', 'operator', 'charset',)

# 'LocationRaw' has a lot of data with only Country or a Direction,
# so we need to filter the data to get the most accurate data
# here i set the capital if only the country is in the data
```

```
# and set the first city of that Direction in **ENGLAND** if the data is a direction
IGNORE_SIGNS = ['UK', 'GB', 'United Kingdom', 'Great Britain', 'England', 'Scotland', 'Wales', 'Northern Ir
IGNORE_ENGLAND = ['North England',
                  'North West England', 'North East England',
                  'South England',
                  'South West England', 'South East England',
                  'East England', 'East of England',
                  'West England', 'West of England',
                  'Midlands',
                  'East Midlands', 'West Midlands']
IGNORE_DIRECTIONS = ['North', 'South', 'East', 'West',
                    'North West', 'North East',
                    'South West', 'South East']
```

In [834…

```
class LocationRetrieval():
    def __init__(self, low_request_dict):
        #save a search result to directly use so less api request
        self.low_request_dict = low_request_dict

    def fetchJson(self, method, params):
        uri = DOMAIN + '%s?%s&username=%s' % (method, urllib.parse.urlencode(params), USERNAME)
        resource = urllib.request.urlopen(uri).readlines()
        js = json.loads(resource[0])
        return js

    #this is the code for the api request
    def search(self, **kwargs):
        method = 'searchJSON'
        valid_kwargs = VALID_KWARGS
        params = {}
        custom_params = {'country': 'GB', 'maxRows': 1, 'lang': 'en', 'style': 'FULL', 'featureClass': 'P'}
        params.update(custom_params)

        for key in kwargs:
            if key in valid_kwargs:
                params[key] = kwargs[key]

        # fuzzy search mode, include worldwide location
        if kwargs['fuzzy']:
            del params['maxRows']
            del params['country']
            del params['featureClass']
```

```
            params['fuzzy'] = 0.7

        results = self.fetchJson(method, params)
        if('geonames' in results):
            return results['geonames']
        else:
            return None

    #this is the main function to get the result of the location
    def check_search_and_save(self, q, fuzzy=False):

        #check if already searched in past
        if q in self.low_request_dict:
            return (self.low_request_dict[q]['Country'], self.low_request_dict[q]['County'], self.low_reque

        #search
        results = self.search(q=q, fuzzy=fuzzy)
        if not results:
            return False
        else:
            result = results[0]
            if len(results) > 1:
                #get the info of UK's
                for r in results:
                    if r['adminName1'] and r['countryCode'] == 'GB':
                        result = r
                        break

            #fclName means the type of location
            if result['fclName'] == 'parks,area, ...':
                if result['fcodeName'] == 'continent':
                    Country = County = City = -1
                else:
                    Country = result['adminName1']
                    County = City = result['name']
            elif result['fclName'] == 'country, state, region,...':
                Country = County = City = -1
            elif result['fclName'] == 'mountain,hill,rock,... ':
                Country = result['adminName1']
                County = City = result['name']
            else:
                #another country
```

```
                if result['adminName1'] and result['countryCode'] != 'GB':
                    Country = result['countryName']
                    County = result['adminName1']
                    City = result['name']
                #GB situation (best situation)
                else:
                    Country = result['adminName1']
                    County = result['adminName2']
                    City = result['name']

        if Country and County and City:
            #save
            self.low_request_dict[q] = {'Country': Country, 'County': County, 'City': City}
            return (Country, County, City)
        else:
            return False

def is_sign_to_ignore(q, ignores):
    for ignore in ignores:
        if q.lower() == ignore.lower():
            return True
    return False

def contains_london(q):
    pattern = re.compile(r'london', re.IGNORECASE)
    return bool(pattern.search(q))

def contains_keys(q, keys: dict):
    keys = list(keys.keys())
    for key in keys:
        if key.lower() in q.lower():
            return key
    return None

# check if 'LocationRaw' is valid for search
def has_english_letters(input_string):
    return bool(re.search(r'[a-zA-Z]', input_string))
```

In [351…

```
df_location_raw = pd.DataFrame({'LocationRaw': df_train['LocationRaw']})
low_request_dict = {}
```

In [ ]:
```
#save for debug
debug_missing = []

retriver = LocationRetrieval(low_request_dict)

tqdm.pandas()
for index, row in tqdm(df_location_raw.iterrows(), total=len(df_location_raw)):

    #define flags
    flag = False #find correct result
    Missing = False #Original data is incorrect or not in the database
    if not has_english_letters(row['LocationRaw']):
        continue

    elements = row['LocationRaw'].lower()
    elements = re.split(',|/|_|&', elements)
    #1. find from the splited list of str (ignore countries and directions situation)
    if not flag and len(elements) > 1:
        for element in elements:
            q = element.strip()
            if len(q) == 1 or is_sign_to_ignore(q, IGNORE_SIGNS) or is_sign_to_ignore(q, IGNORE_ENGLAND) or
                continue
            if contains_london(q):
                q = 'london'
            #first check
            flag = retriver.check_search_and_save(q)
            if flag:
                break
            #second check with fuzzy
            flag = retriver.check_search_and_save(q, fuzzy=True)
            if flag:
                break

    #2. we include the countries and directions situation to search again
    if not flag:
        for element in elements:
            q = element.strip()
            if contains_keys(q, low_request_dict):
                q = contains_keys(q, low_request_dict)
                flag = retriver.check_search_and_save(q)
            elif is_sign_to_ignore(q, IGNORE_SIGNS):
                flag = retriver.check_search_and_save(q)
```

```
            elif is_sign_to_ignore(q, IGNORE_ENGLAND):
                flag = retriver.check_search_and_save(q, fuzzy=True)
            elif is_sign_to_ignore(q, IGNORE_DIRECTIONS):
                flag = retriver.check_search_and_save(q)

    #3. we open fuzzy search which will search not only UK but worldwide
    if not flag:
        element = elements[0]
        element = element.strip()
        #fuzzy separate search
        elements = re.split(' ', element)
        #check
        for element in elements:
            q = element.strip()
            if contains_keys(q, low_request_dict):
                q = contains_keys(q, low_request_dict)
                flag = retriver.check_search_and_save(q)
                break
        #search
        if not flag:
            for element in elements:
                q = element.strip()
                flag = retriver.check_search_and_save(q)
                if flag:
                    break

    if flag:
        df_location_raw.loc[index, 'Country'] = flag[0]
        df_location_raw.loc[index, 'County'] = flag[1]
        df_location_raw.loc[index, 'City'] = flag[2]

    if not flag:
        print(index)
        debug_missing.append(index)
```

In [847…

```
#save for next use
with open('./data/Location_output.json', 'w') as json_file:
    json.dump(low_request_dict, json_file, indent=2)
```

In [908…

```
df_location_raw.replace(-1, np.nan, inplace=True)
df_location_raw = df_location_raw.drop('LocationRaw', axis=1)
```

```
df_location_raw.to_csv('Train_location.csv', index=False)
```

Missing Values

In [ ]:
```
#merge with our calibrated location information
df_train_with_location = pd.concat([df_train, df_location_raw], axis=1)

#reshape columns order
columns_index = ['Title', 'FullDescription', 'Company', 'Country', 'County', 'City', 'ContractTime', 'Categ
df_train_ordered = df_train_with_location[columns_index]

#fill missing data with 'unknown' for 'ContractTime'
df_train_ordered.loc[:,'ContractTime'].replace(np.nan, "unknown", inplace=True)
```

In [140…
```
df_train_ordered.to_csv('Train_Missing_Fixed.csv', index=False)
```

Text combine and cleaning

In [8]:
```
def concatenate_fields(row):
    return (f"{row['Title']}\n"
            f"Location: {row['Country']}, {row['County']}, {row['City']}\n"
            f"Contract Time: {row['ContractTime']}\n"
            f"Company: {row['Company']}\n"
            f"Category: {row['Category']}\n"
            f"Description:\n{row['FullDescription']}\n")

def remove_end(txt):
    txt = txt[:txt.rfind('-')]
    return re.sub('-.+?, [0-9]+','',txt)

def remove_urls(txt):
    txt = re.sub(r'html\S+','',txt)
    txt = re.sub(r'http\S+','',txt)
    txt = re.sub(r'pic.\S+','',txt)
    txt = re.sub(r'www.\S+','',txt)
    return txt

def remove_escaped(txt):
    return re.sub(r'&\S+','',txt)
```

```
def text_preprocess(txt):
    text_process_list = [remove_escaped, remove_urls, remove_end]
    txt = reduce(lambda txt,func: func(txt),text_process_list,txt)
    return txt

def get_processed_df(df):
    for i, row in df.iterrows():
        full_text = concatenate_fields(row)
        full_text = text_preprocess(full_text)
        df.loc[i, 'FullText'] = full_text
    return df
```

In [9]:
```
df_train_processed = get_processed_df(df_train_ordered)
```

In [10]:
```
df_train_processed.to_csv('Train_Text_Processed.csv', index=False)
```

Embed text

In [12]:
```
from sentence_transformers import SentenceTransformer
import torch
import torch.nn as nn

def get_device_and_set_seed(seed):
    """ Set all seeds to make results reproducible """
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    np.random.seed(seed)
    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda:0" if use_cuda else "cpu")
    return device

SEED = 123
device = get_device_and_set_seed(SEED)

sentence_model_name = 'MohammedDhiyaEddine/job-skill-sentence-transformer-tsdae'
sentence_model = SentenceTransformer(sentence_model_name).to(device)
```

In [ ]:
```
#get text embedding from pretrained model
def get_sentence_embedding(model, sentences):
```

```
    batch_size = 128
    for batch in tqdm(range(0, len(sentences), batch_size)):
        if batch + batch_size < len(sentences):
            X_text_batch = sentences[batch:batch+batch_size]
        else:
            X_text_batch = sentences[batch:]

        if batch == 0:
            X_text_emb = model.encode(X_text_batch, convert_to_tensor=True)
        else:
            X_text_emb = torch.cat((X_text_emb, model.encode(X_text_batch, convert_to_tensor=True)), 0)

    return X_text_emb
```

In [8]:
```
text_embeddings = get_sentence_embedding(sentence_model, df_train_processed['FullText'].tolist())
```

```
100%|██████████| 244768/244768 [30:06<00:00, 135.50it/s]
```

Modeling

In [7]:
```
'''
Design:
1. simple NN from text embeddings to salary
2. split salary to classes and use a classification model with cross entropy loss
3.
'''
text_embeddings = torch.load('/kaggle/input/ml-project-dataset/text_embed.pt')
df = pd.read_csv('/kaggle/input/job-salary-predict/Train_Text_Processed.csv')
targets = df[['SalaryNormalized']]

#Divide the original salary into k intervals
salary_range = list(range(5000, 200001, 500))
print(f'We gonna have {len(salary_range)} classes for salary.')

#assign the closest interval to each sample
salary_range_dict = {value: index for index, value in enumerate(salary_range)}
salary_range_dict_reverse = {index: value for index, value in enumerate(salary_range)}
def map_to_nearest_range(salary):
    return salary_range_dict[min(salary_range, key=lambda x: abs(x - salary))]

targets.loc[:, 'mapped_salary'] = targets['SalaryNormalized'].apply(map_to_nearest_range)
```

```
We gonna have 391 classes for salary.
/tmp/ipykernel_34/4193637675.py:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
#returning-a-view-versus-a-copy
  targets.loc[:, 'mapped_salary'] = targets['SalaryNormalized'].apply(map_to_nearest_range)
```

In [8]:
```
#Split
X_train, X_test, df_y_train, df_y_test = train_test_split(text_embeddings, targets, test_size=0.2, random_s
```

In [19]:
```
#put it into tensor
y_train = torch.from_numpy(df_y_train['SalaryNormalized'].values).long().to(device)
y_test = torch.from_numpy(df_y_test['SalaryNormalized'].values).long().to(device)
```

In [20]:
```
class Model(nn.Module):
    def __init__(self):
        super().__init__()
```

```
        self.regression = nn.Sequential(
            nn.Linear(768, 1536),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(1536, 768),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(768, 384),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(384, 384),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(384, 1)
        )

    def forward(self, x):
        return self.regression(x)

model = Model().to(device)
```

In [23]:
```
mse = nn.MSELoss()
mae = nn.L1Loss()
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)

epochs = 300
batch_size = 128

lowest_mae = 10000000
```

In [24]:
```
for epoch in range(epochs):
    model.train()
    for batch in range(0, len(X_train), batch_size):
        optimizer.zero_grad()

        inputs = X_train[batch:batch+batch_size]
        labels = y_train[batch:batch+batch_size]

        output = model(inputs)
        loss = mse(output.squeeze(), labels.float())
```

```python
        loss.backward()
        optimizer.step()

    model.eval()
    with torch.no_grad():
        output = model(X_test)
        loss = mae(output.squeeze(), y_test.float())
        if loss < lowest_mae:
            lowest_mae = loss
            print('--------------------------------------------------')
            print(f'MAE: {loss}, Epoch: {epoch}')
            print('--------------------------------------------------')
```

```
--------------------------------------------------
MAE: 9981.1943359375, Epoch: 0
--------------------------------------------------
--------------------------------------------------
MAE: 9410.0927734375, Epoch: 1
--------------------------------------------------
--------------------------------------------------
MAE: 8991.494140625, Epoch: 2
--------------------------------------------------
--------------------------------------------------
MAE: 8811.890625, Epoch: 3
--------------------------------------------------
--------------------------------------------------
MAE: 8627.8994140625, Epoch: 4
--------------------------------------------------
--------------------------------------------------
MAE: 8490.45703125, Epoch: 5
--------------------------------------------------
--------------------------------------------------
MAE: 8384.01953125, Epoch: 6
--------------------------------------------------
--------------------------------------------------
MAE: 8304.2470703125, Epoch: 7
--------------------------------------------------
--------------------------------------------------
MAE: 8232.9091796875, Epoch: 8
--------------------------------------------------
--------------------------------------------------
MAE: 8143.30322265625, Epoch: 9
--------------------------------------------------
--------------------------------------------------
MAE: 8064.83544921875, Epoch: 10
--------------------------------------------------
--------------------------------------------------
MAE: 7982.14453125, Epoch: 11
--------------------------------------------------
--------------------------------------------------
MAE: 7920.2060546875, Epoch: 12
--------------------------------------------------
--------------------------------------------------
MAE: 7868.17041015625, Epoch: 13
--------------------------------------------------
```

```
---------------------------------------------------
MAE: 7807.25830078125, Epoch: 14
---------------------------------------------------
---------------------------------------------------
MAE: 7765.4052734375, Epoch: 15
---------------------------------------------------
---------------------------------------------------
MAE: 7715.1787109375, Epoch: 16
---------------------------------------------------
---------------------------------------------------
MAE: 7697.927734375, Epoch: 17
---------------------------------------------------
---------------------------------------------------
MAE: 7648.845703125, Epoch: 18
---------------------------------------------------
---------------------------------------------------
MAE: 7576.84619140625, Epoch: 19
---------------------------------------------------
---------------------------------------------------
MAE: 7560.11083984375, Epoch: 21
---------------------------------------------------
---------------------------------------------------
MAE: 7543.3271484375, Epoch: 22
---------------------------------------------------
---------------------------------------------------
MAE: 7519.14697265625, Epoch: 23
---------------------------------------------------
---------------------------------------------------
MAE: 7506.18994140625, Epoch: 24
---------------------------------------------------
---------------------------------------------------
MAE: 7427.63330078125, Epoch: 27
---------------------------------------------------
---------------------------------------------------
MAE: 7400.7548828125, Epoch: 28
---------------------------------------------------
---------------------------------------------------
MAE: 7383.16650390625, Epoch: 30
---------------------------------------------------
---------------------------------------------------
MAE: 7362.66455078125, Epoch: 31
---------------------------------------------------
```

```
--------------------------------------------------
MAE: 7325.7763671875, Epoch: 32
--------------------------------------------------
--------------------------------------------------
MAE: 7254.67138671875, Epoch: 37
--------------------------------------------------
--------------------------------------------------
MAE: 7130.1103515625, Epoch: 38
--------------------------------------------------
--------------------------------------------------
MAE: 7127.86328125, Epoch: 41
--------------------------------------------------
--------------------------------------------------
MAE: 7124.1435546875, Epoch: 42
--------------------------------------------------
--------------------------------------------------
MAE: 7120.6025390625, Epoch: 43
--------------------------------------------------
--------------------------------------------------
MAE: 7100.05908203125, Epoch: 44
--------------------------------------------------
--------------------------------------------------
MAE: 7065.4365234375, Epoch: 45
--------------------------------------------------
--------------------------------------------------
MAE: 7007.76806640625, Epoch: 46
--------------------------------------------------
--------------------------------------------------
MAE: 6984.1748046875, Epoch: 48
--------------------------------------------------
--------------------------------------------------
MAE: 6979.875, Epoch: 50
--------------------------------------------------
--------------------------------------------------
MAE: 6910.0546875, Epoch: 52
--------------------------------------------------
--------------------------------------------------
MAE: 6865.75634765625, Epoch: 54
--------------------------------------------------
--------------------------------------------------
MAE: 6837.8583984375, Epoch: 58
--------------------------------------------------
```

```
--------------------------------------------------
MAE: 6789.4482421875, Epoch: 59
--------------------------------------------------
--------------------------------------------------
MAE: 6771.361328125, Epoch: 62
--------------------------------------------------
--------------------------------------------------
MAE: 6755.80712890625, Epoch: 63
--------------------------------------------------
--------------------------------------------------
MAE: 6741.25244140625, Epoch: 66
--------------------------------------------------
--------------------------------------------------
MAE: 6732.2119140625, Epoch: 67
--------------------------------------------------
--------------------------------------------------
MAE: 6702.90771484375, Epoch: 73
--------------------------------------------------
--------------------------------------------------
MAE: 6650.3681640625, Epoch: 77
--------------------------------------------------
--------------------------------------------------
MAE: 6641.76513671875, Epoch: 83
--------------------------------------------------
--------------------------------------------------
MAE: 6605.75927734375, Epoch: 84
--------------------------------------------------
--------------------------------------------------
MAE: 6605.5400390625, Epoch: 86
--------------------------------------------------
--------------------------------------------------
MAE: 6563.5810546875, Epoch: 87
--------------------------------------------------
--------------------------------------------------
MAE: 6534.994140625, Epoch: 89
--------------------------------------------------
--------------------------------------------------
MAE: 6517.822265625, Epoch: 96
--------------------------------------------------
--------------------------------------------------
MAE: 6510.083984375, Epoch: 103
--------------------------------------------------
```

```
--------------------------------------------------
MAE: 6476.5205078125, Epoch: 115
--------------------------------------------------
--------------------------------------------------
MAE: 6455.5439453125, Epoch: 119
--------------------------------------------------
--------------------------------------------------
MAE: 6417.05224609375, Epoch: 121
--------------------------------------------------
--------------------------------------------------
MAE: 6400.92041015625, Epoch: 151
--------------------------------------------------
--------------------------------------------------
MAE: 6344.78955078125, Epoch: 156
--------------------------------------------------
--------------------------------------------------
MAE: 6318.81103515625, Epoch: 162
--------------------------------------------------
--------------------------------------------------
MAE: 6276.9169921875, Epoch: 187
--------------------------------------------------
--------------------------------------------------
MAE: 6247.46484375, Epoch: 189
--------------------------------------------------
--------------------------------------------------
MAE: 6223.40576171875, Epoch: 202
--------------------------------------------------
--------------------------------------------------
MAE: 6209.93505859375, Epoch: 207
--------------------------------------------------
--------------------------------------------------
MAE: 6168.009765625, Epoch: 213
--------------------------------------------------
--------------------------------------------------
MAE: 6154.5322265625, Epoch: 274
--------------------------------------------------
--------------------------------------------------
MAE: 6152.86572265625, Epoch: 275
--------------------------------------------------
--------------------------------------------------
MAE: 6151.18896484375, Epoch: 276
--------------------------------------------------
```

```
-------------------------------------------------
MAE: 6113.9638671875, Epoch: 278
-------------------------------------------------
-------------------------------------------------
MAE: 6090.01318359375, Epoch: 282
-------------------------------------------------
-------------------------------------------------
MAE: 6050.2080078125, Epoch: 284
-------------------------------------------------
-------------------------------------------------
MAE: 6033.84130859375, Epoch: 286
-------------------------------------------------
-------------------------------------------------
MAE: 6011.24658203125, Epoch: 287
-------------------------------------------------
-------------------------------------------------
MAE: 5998.39794921875, Epoch: 298
-------------------------------------------------
```

In [25]:

```
#save model

torch.save(model.state_dict(), 'model_state_dict.pth')
```

In [27]:

```
y_train = torch.from_numpy(df_y_train['mapped_salary'].values).long().to(device)
y_test = torch.from_numpy(df_y_test['SalaryNormalized'].values).long().to(device)
```

In [40]:

```
class Model_Mapped(nn.Module):
    def __init__(self):
        super().__init__()

        self.classification = nn.Sequential(
            nn.Linear(768, 3072),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(3072, 1536),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(1536, 768),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(768, 768),
```

```
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(768, 391)
        )

    def forward(self, x):
        return self.classification(x)

model_mapped = Model_Mapped().to(device)
```

In [41]:
```
cross_entropy = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(model_mapped.parameters(), lr=5e-5)

epochs = 300
batch_size = 128

lowest_mae = 10000000
```

In [ ]:
```
for epoch in range(epochs):
    model_mapped.train()
    for batch in range(0, len(X_train), batch_size):
        optimizer.zero_grad()

        inputs = X_train[batch:batch+batch_size]
        labels = y_train[batch:batch+batch_size]

        output = model_mapped(inputs)
        loss = cross_entropy(output, labels)
        loss.backward()
        optimizer.step()

    model_mapped.eval()
    with torch.no_grad():
        output = model_mapped(X_test)
        preds = output.argmax(dim=1)
        preds_value = torch.tensor([salary_range_dict_reverse[key.item()] for key in preds]).to(device)
        loss = mae(preds_value, y_test.float())
        if loss < lowest_mae:
            lowest_mae = loss
            print('--------------------------------------------------')
            print(f'MAE: {loss}, Epoch: {epoch}')
```

```
        print('--------------------------------------------------')
```

```
--------------------------------------------------
MAE: 7875.09130859375, Epoch: 0
--------------------------------------------------
--------------------------------------------------
MAE: 7715.4267578125, Epoch: 1
--------------------------------------------------
--------------------------------------------------
MAE: 7493.73681640625, Epoch: 2
--------------------------------------------------
--------------------------------------------------
MAE: 7482.83203125, Epoch: 3
--------------------------------------------------
--------------------------------------------------
MAE: 7383.83447265625, Epoch: 4
--------------------------------------------------
--------------------------------------------------
MAE: 7246.47509765625, Epoch: 5
--------------------------------------------------
--------------------------------------------------
MAE: 7169.50048828125, Epoch: 6
--------------------------------------------------
--------------------------------------------------
MAE: 7017.37890625, Epoch: 7
--------------------------------------------------
--------------------------------------------------
MAE: 6854.94580078125, Epoch: 10
--------------------------------------------------
--------------------------------------------------
MAE: 6829.29443359375, Epoch: 12
--------------------------------------------------
--------------------------------------------------
MAE: 6761.75634765625, Epoch: 14
--------------------------------------------------
--------------------------------------------------
MAE: 6684.90673828125, Epoch: 15
--------------------------------------------------
--------------------------------------------------
MAE: 6629.537109375, Epoch: 18
--------------------------------------------------
--------------------------------------------------
MAE: 6595.4638671875, Epoch: 22
--------------------------------------------------
```

```
--------------------------------------------------
MAE: 6590.76611328125, Epoch: 31
--------------------------------------------------
--------------------------------------------------
MAE: 6394.06201171875, Epoch: 33
--------------------------------------------------
--------------------------------------------------
MAE: 6310.15234375, Epoch: 34
--------------------------------------------------
--------------------------------------------------
MAE: 6290.08935546875, Epoch: 44
--------------------------------------------------
--------------------------------------------------
MAE: 6201.66650390625, Epoch: 45
--------------------------------------------------
--------------------------------------------------
MAE: 6192.4384765625, Epoch: 46
--------------------------------------------------
--------------------------------------------------
MAE: 6110.10107421875, Epoch: 47
--------------------------------------------------
--------------------------------------------------
MAE: 6073.44775390625, Epoch: 50
--------------------------------------------------
--------------------------------------------------
MAE: 6033.30419921875, Epoch: 53
--------------------------------------------------
```

In [ ]:

```
torch.save(model_mapped.state_dict(), 'model_mapped_state_dict.pth')
```