Kaggle Competition - Season 4 Episode 11

Initial look at data

In [186…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [187…
```
train = pd.read_csv('./train.csv')
test = pd.read_csv('./test.csv')
og = pd.read_csv('./final_depression_dataset_1.csv')
ids = test['id']
train.head()
```

Out[187…

	id	Name	Gender	Age	City	Working Professional or Student	Profession	Academic Pressure	W Pres
0	0	Aaradhya	Female	49.0	Ludhiana	Working Professional	Chef	NaN	
1	1	Vivan	Male	26.0	Varanasi	Working Professional	Teacher	NaN	
2	2	Yuvraj	Male	33.0	Visakhapatnam	Student	NaN	5.0	
3	3	Yuvraj	Male	22.0	Mumbai	Working Professional	Teacher	NaN	
4	4	Rhea	Female	30.0	Kanpur	Working Professional	Business Analyst	NaN	

In [188…
```
print(f'Size of train data: {train.shape}')
print(f'Size of test data: {test.shape}')
print(f'Size of og data: {og.shape}')
```

```
Size of train data: (140700, 20)
Size of test data: (93800, 19)
Size of og data: (2556, 19)
```

Checking for unique values for every feature

In [189…
```
unique_vals_df = pd.DataFrame({'train': train.nunique(), 'test': test.nunique(),
unique_vals_df
```

Out[189…

	train	test	og
Academic Pressure	5	5.0	5.0
Age	43	44.0	43.0
CGPA	331	326.0	312.0
City	98	68.0	30.0
Degree	115	87.0	27.0
Depression	2	NaN	2.0
Dietary Habits	23	22.0	3.0
Family History of Mental Illness	2	2.0	2.0
Financial Stress	5	5.0	5.0
Gender	2	2.0	2.0
Have you ever had suicidal thoughts ?	2	2.0	2.0
Job Satisfaction	5	5.0	5.0
Name	422	374.0	216.0
Profession	64	64.0	35.0
Sleep Duration	36	31.0	4.0
Study Satisfaction	5	5.0	5.0
Work Pressure	5	5.0	5.0
Work/Study Hours	13	13.0	13.0
Working Professional or Student	2	2.0	2.0
id	140700	93800.0	NaN

In [190…

```
dropping_cols = ['id','Name', 'Academic Pressure', 'CGPA', 'Study Satisfaction']
# Check if columns exist in the dataframe before dropping them to safegaurd agai
dropping_cols = [col for col in dropping_cols if col in train.columns]
train.drop(dropping_cols, inplace=True, axis=1)
test.drop(dropping_cols, inplace=True, axis=1)
og.drop(dropping_cols[1:], inplace=True, axis=1)
```

Create a copy of the dataframe

In [191…

```
temp_og = og.copy()
temp_test = test.copy()
temp_train = train.copy()
```

In [192…

```
train.isnull().sum()
```

Out[192…

```
Gender                                      0
Age                                         0
City                                        0
Working Professional or Student             0
Profession                              36630
Work Pressure                           27918
Job Satisfaction                        27910
Sleep Duration                              0
Dietary Habits                              4
Degree                                      2
Have you ever had suicidal thoughts ?       0
Work/Study Hours                            0
Financial Stress                            4
Family History of Mental Illness            0
Depression                                  0
dtype: int64
```

Explore the data

Clean features

For degree

In [193…

```
import re

# We need to clean this list
def clean_string(input_string):
    return re.sub(r'[^a-z0-9]', '', str(input_string).lower())

og_degree_list = og['Degree'].unique()
og_degree_list = [clean_string(x) for x in og_degree_list]
```

In [194…

```
# Now, I will first clean the Degree column in the train and test dataframes
train['Degree'] = train['Degree'].apply(lambda x: clean_string(x))
test['Degree'] = test['Degree'].apply(lambda x: clean_string(x))
og['Degree'] = og['Degree'].apply(lambda x: clean_string(x))

#Now, I will replace the values in the Degree column in the train and test dataf
train['Degree'] = train['Degree'].apply(lambda x: x if x in og_degree_list else
test['Degree'] = test['Degree'].apply(lambda x: x if x in og_degree_list else No
og['Degree'] = og['Degree'].apply(lambda x: x if x in og_degree_list else None)
```

For Profession

In [195…

```
og_profession_list = set(og['Profession'].unique())
og_profession_list = {clean_string(x) for x in og_profession_list}
og_profession_list.remove('nan')

# Now, I will first clean the Profession column in the train and test dataframes
train['Profession'] = train['Profession'].apply(lambda x: clean_string(x))
test['Profession'] = test['Profession'].apply(lambda x: clean_string(x))
og['Profession'] = og['Profession'].apply(lambda x: clean_string(x))

#Now, I will replace the values in the Profession column in the train and test d
train['Profession'] = train['Profession'].apply(lambda x: x if x in og_professio
```

```
test['Profession'] = test['Profession'].apply(lambda x: x if x in og_profession_
og['Profession'] = og['Profession'].apply(lambda x: x if x in og_profession_list
```

Age

In [196…

```
# Convert the 'Age' column in the 'train' dataframe to integer type
train['Age'] = train['Age'].astype(int)
test['Age'] = test['Age'].astype(int)
og['Age'] = og['Age'].astype(int)
temp_train['Age'] = temp_train['Age'].astype(int)
```

Work/Study hours

In [197…

```
train['Work/Study Hours'] = train['Work/Study Hours'].astype(int)
test['Work/Study Hours'] = test['Work/Study Hours'].astype(int)
og['Work/Study Hours'] = og['Work/Study Hours'].astype(int)
```

City

In [198…

```
#Lets check the city now
#Since we reliably know that the citys in og dataset are correct, we check for t

# exclusive_train_city_list = train['City'].unique()[~np.isin(train['City'].uniq
# exclusive_test_city_list = test['City'].unique()[~np.isin(test['City'].unique(
#We didnt include them since they were mostly fabricated
final_city_list = set(og['City'].unique()).union(['Gurgaon'])

#Now, I will first clean the City column in the train and test dataframes
train['City'] = train['City'].apply(lambda x: x if x in final_city_list else Non
test['City'] = test['City'].apply(lambda x: x if x in final_city_list else None)
og['City'] = og['City'].apply(lambda x: x if x in final_city_list else None)
```

Sleep duration

In [199…

```
og_sleep_list = og['Sleep Duration'].unique()

#Now, I will replace the values in the Sleep Duration column in the train and te
train['Sleep Duration'] = train['Sleep Duration'].apply(lambda x: x if x in og_s
test['Sleep Duration'] = test['Sleep Duration'].apply(lambda x: x if x in og_sle
```

Dietary Habits

In [200…

```
#Now, I will replace the values in the Sleep Duration column in the train and te

og_diet_list = og['Dietary Habits'].unique()

#Now, I will replace the values in the Dietary Habits column in the train and te
train['Dietary Habits'] = train['Dietary Habits'].apply(lambda x: x if x in og_d
test['Dietary Habits'] = test['Dietary Habits'].apply(lambda x: x if x in og_die
```

Coverting binary cat cols to boolean

In [201…

```
train.rename(columns={'Have you ever had suicidal thoughts ?' : 'Suicidal Though

test.rename(columns={'Have you ever had suicidal thoughts ?' : 'Suicidal Thought
```

```
og.rename(columns={'Have you ever had suicidal thoughts ?' : 'Suicidal Thoughts'
```

In [202…

```
#Now convert binary cat cols to 0 and 1
train['Suicidal Thoughts'] = train['Suicidal Thoughts'].apply(lambda x: 1 if x =
test['Suicidal Thoughts'] = test['Suicidal Thoughts'].apply(lambda x: 1 if x ==
og['Suicidal Thoughts'] = og['Suicidal Thoughts'].apply(lambda x: 1 if x == 'Yes

train['is_male'] = train['is_male'].apply(lambda x : 1 if x == 'Male' else 0)
test['is_male'] = test['is_male'].apply(lambda x : 1 if x == 'Male' else 0)
og['is_male'] = og['is_male'].apply(lambda x : 1 if x == 'Male' else 0)

train['is_student'] = train['is_student'].apply(lambda x : 1 if x == 'Student' e
test['is_student'] = test['is_student'].apply(lambda x : 1 if x == 'Student' els
og['is_student'] = og['is_student'].apply(lambda x : 1 if x == 'Student' else 0)

train['history_mental_illness'] = train['history_mental_illness'].apply(lambda x
test['history_mental_illness'] = test['history_mental_illness'].apply(lambda x :
og['history_mental_illness'] = og['history_mental_illness'].apply(lambda x : 1 i
```

Missing values

In [203…

```
#Dataframe of missing values
#Checking missing values
missing_vals_df = pd.DataFrame({'train': train.isnull().sum().loc[train.isnull()
                                'test': test.isnull().sum().loc[test.isnull().su
                                'og': og.isnull().sum().loc[og.isnull().sum()> 0
                                })

missing_vals_df
```

Out[203…

	train	test	og
City	97	50.0	NaN
Degree	101	67.0	NaN
Dietary Habits	27	30.0	NaN
Financial Stress	4	NaN	NaN
Job Satisfaction	27910	18774.0	502.0
Profession	36680	24676.0	673.0
Sleep Duration	79	54.0	NaN
Work Pressure	27918	18778.0	502.0

Check the nature of these cols

- Degree - Categorical
- Profession - Categorical
- Work pressure - Numerical
- Job satisfaction - Numerical
- Dietary Habits - Categorical
- Financial Stress - Numerical

Redundancy to check for enough records after dropping missing values

In [204…
```
#Check whether there are enough records to train the model
# train_without_names = train.drop(['Name', 'id'], axis=1)
# number_of_duplicates = train_without_names.duplicated().sum()
```

Drop missing value entries with low frequency

Degree, Dietary Habits, Financial Stress have low missing values. These can be dropped.

In [205…
```
low_missing_cols = ['Degree', 'Dietary Habits', 'Financial Stress', 'City', 'Sle
low_missing_cols_train = [col for col in low_missing_cols if col in train.column
low_missing_cols_test = [col for col in low_missing_cols if col in train.columns
train = train.dropna(subset=low_missing_cols)
test = test.dropna(subset=low_missing_cols)
```

Dealing with missing value entries with high frequency

In [206…
```
#Derive high missing frequency cols
high_missing_cols= ['Work Pressure', 'Job Satisfaction', 'Profession']
datasets_to_vizualize = ['train', 'test', 'og']

#There are three high_missing_cols_num and two datasets to visualize
#I want to visualize the distribution in the three datasets
# fig,ax = plt.subplots(2, 3, figsize=(15, 10))
# fig.suptitle('Distribution of high_missing_cols_num')
# # Iterate over each column and dataset
# for i, col in enumerate(high_missing_cols[:2]):
#     for j, dataset in enumerate(datasets_to_vizualize):
#         # plt.title(f'Distribution of {col} in {dataset}')
#         ax[i, j].set_title(f'Distribution of {col} in {dataset}')
#         df = eval(dataset)
#         sns.barplot(data = df, x = df[col].value_counts().index, y = df[col].v
# plt.tight_layout()
# plt.savefig('graphs/wp_and_js_before.jpg', dpi=300)
# plt.show()
```

Creating a Stratified Proportional Imputer

In [207…
```
#Use random sampling imputation to fill the missing values in the high_missing_c
#For that, we will make a StratifiedProportionalImputer class

from sklearn.base import BaseEstimator, TransformerMixin
import warnings

class StratifiedProportionalImputer(BaseEstimator, TransformerMixin):
    """
    A custom transformer to impute missing values in a feature while preserving
    the proportionality and distribution of the non-missing values.

    Parameters:
    ----------
    random_state : int, optional
        Random seed for reproducibility.
    """
    def __init__(self, random_seed = 42):
```

```
        self.random_seed = random_seed
        warnings.warn('Use this only for categorical values!', UserWarning)

    def fit(self, dataframe:pd.DataFrame, imputation_cols: list[str]):
        """
        Fit the transformer by computing the observed value distributions for ca

        Parameters:
        ----------
        dataframe : pd.DataFrame
            The input dataframe.
        y : None
            Ignored, for compatibility with sklearn pipelines.

        Returns:
        -------
        self : StratifiedProportionalImputer
            The fitted transformer.
        """
        if not isinstance(dataframe, pd.DataFrame):
            raise ValueError("Input must be a pandas DataFrame.")

        self.proportional_rations_ = {}
        self.rng = np.random.default_rng(self.random_seed)
        df = dataframe.copy()
        no_null_df = dataframe.copy().dropna(subset=imputation_cols)

        for col in imputation_cols:
            temp_dict = no_null_df[col].value_counts().to_dict()
            total = sum(temp_dict.values())
            self.proportional_rations_[col] = {k: v/total for k, v in temp_dict.
        pass

    def fit_transform(self, dataframe:pd.DataFrame, imputation_cols: list[str]):
        """
        Fit the transformer by computing the observed value distributions for ca

        Parameters:
        ----------
        dataframe : pd.DataFrame
            The input dataframe.
        y : None
            Ignored, for compatibility with sklearn pipelines.

        Returns:
        -------
        self : StratifiedProportionalImputer
            The fitted transformer.
        """
        if not isinstance(dataframe, pd.DataFrame):
            raise ValueError("Input must be a pandas DataFrame.")

        self.proportional_rations_ = {}
        self.rng = np.random.default_rng(self.random_seed)
        df = dataframe.copy()
        no_null_df = dataframe.copy().dropna(subset=imputation_cols)

        for col in imputation_cols:
            temp_dict = no_null_df[col].value_counts().to_dict()
            total = sum(temp_dict.values())
```

```
            self.proportional_rations_[col] = {k: v/total for k, v in temp_dict.

        #Now, we will impute the missing values
        for col in imputation_cols:
            missing_mask = df[col].isnull()
            if missing_mask.sum().any():
                df.loc[missing_mask, col] = self.rng.choice(
                    list(self.proportional_rations_[col].keys()),
                    size = missing_mask.sum(),
                    replace = True,
                    p = list(self.proportional_rations_[col].values())
                )

        return df

    def transform(self, dataframe:pd.DataFrame, imputation_cols: list[str]):

        if not isinstance(dataframe, pd.DataFrame):
            raise ValueError("Input must be a pandas DataFrame.")

        df = dataframe.copy()

        for col in imputation_cols:
            missing_mask = df[col].isnull()
            if missing_mask.sum().any():
                df.loc[missing_mask, col] = self.rng.choice(
                    list(self.proportional_rations_[col].keys()),
                    size = missing_mask.sum(),
                    replace = True,
                    p = list(self.proportional_rations_[col].values())
                )

        return df
```

Implementing and testing the imputer

In [208…

```
#Lets try to impute the missing values in high_missing_cols
imputer = StratifiedProportionalImputer()
#For now, I am including profession, if later it doesnt pan out, I'll deal with
og = imputer.fit_transform(og, ['Work Pressure', 'Job Satisfaction', 'Profession
train = imputer.transform(train, ['Work Pressure', 'Job Satisfaction', 'Professi
test = imputer.transform(test, ['Work Pressure', 'Job Satisfaction', 'Profession
```

```
C:\Users\Acer\AppData\Local\Temp\ipykernel_9068\1767728745.py:19: UserWarning: Us
e this only for categorical values!
  warnings.warn('Use this only for categorical values!', UserWarning)
```

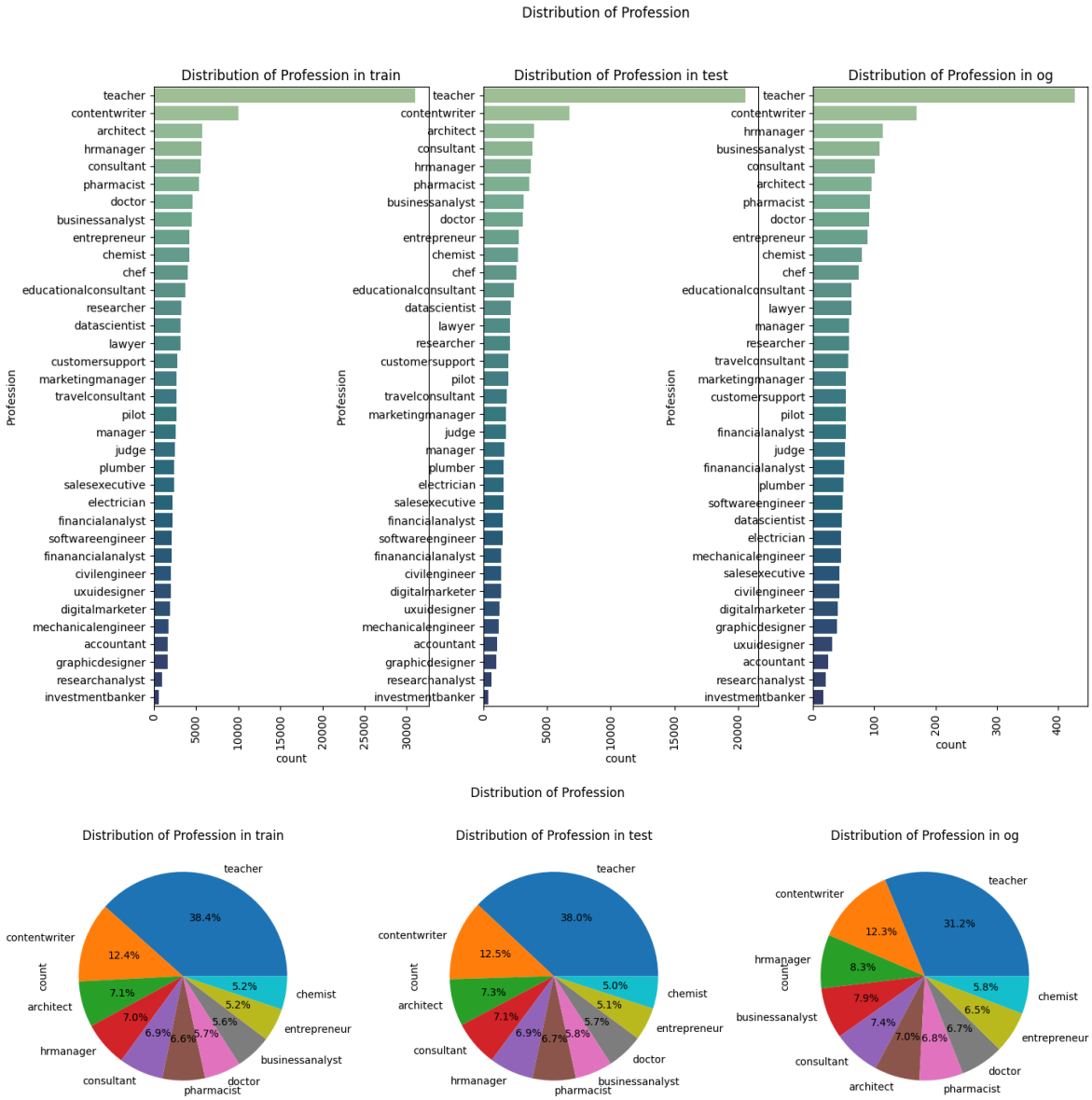
In [209…

```
# fig,ax = plt.subplots(2, 3, figsize=(15, 10))
# fig.suptitle('Distribution of high_missing_cols_num')
# # Iterate over each column and dataset
# for i, col in enumerate(high_missing_cols[:2]):
#     for j, dataset in enumerate(datasets_to_vizualize):
#         # plt.title(f'Distribution of {col} in {dataset}')
#         ax[i, j].set_title(f'Distribution of {col} in {dataset}')
#         df = eval(dataset)
#         sns.barplot(data = df, x = df[col].value_counts().index, y = df[col].v
# plt.tight_layout()
# plt.savefig('graphs/wp_and_js_after.jpg', dpi=300)
# plt.show()
```

In [210…

```
#Now, I will try to visualize the distribution of the 'Profession' column in the
#Since the 'Profession' column has 36 unique values, I will have to use some kin

#Let's try a bar plot
fig, ax = plt.subplots(1, 3, figsize=(15, 10))
fig.suptitle('Distribution of Profession')
for i, dataset in enumerate(datasets_to_vizualize):
    ax[i].set_title(f'Distribution of Profession in {dataset}')
    df = eval(dataset)
    sns.countplot(data=df, y='Profession', ax=ax[i], palette='crest', order=df['
    ax[i].tick_params(axis='x', rotation=90)
plt.savefig('graphs/profession_countplot.jpg', dpi=300)
#Lets try a pie chart
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('Distribution of Profession')
for i, dataset in enumerate(datasets_to_vizualize):
    ax[i].set_title(f'Distribution of Profession in {dataset}')
    df = eval(dataset)
    df['Profession'].value_counts().head(10).plot.pie(autopct='%1.1f%%', ax=ax[i
plt.tight_layout()
plt.savefig('graphs/profession_piechart.jpg', dpi=300)
```

```
C:\Users\Acer\AppData\Local\Temp\ipykernel_9068\2504705095.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(data=df, y='Profession', ax=ax[i], palette='crest', order=df['Pro
fession'].value_counts().index)
C:\Users\Acer\AppData\Local\Temp\ipykernel_9068\2504705095.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(data=df, y='Profession', ax=ax[i], palette='crest', order=df['Pro
fession'].value_counts().index)
C:\Users\Acer\AppData\Local\Temp\ipykernel_9068\2504705095.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(data=df, y='Profession', ax=ax[i], palette='crest', order=df['Pro
fession'].value_counts().index)
```



Exploring numerical data

In [211…

```
#Cleaning and exploring the data
num_cols = ['Age']
null_val = train[num_cols].isnull().sum()
train[num_cols].sample(5)
```

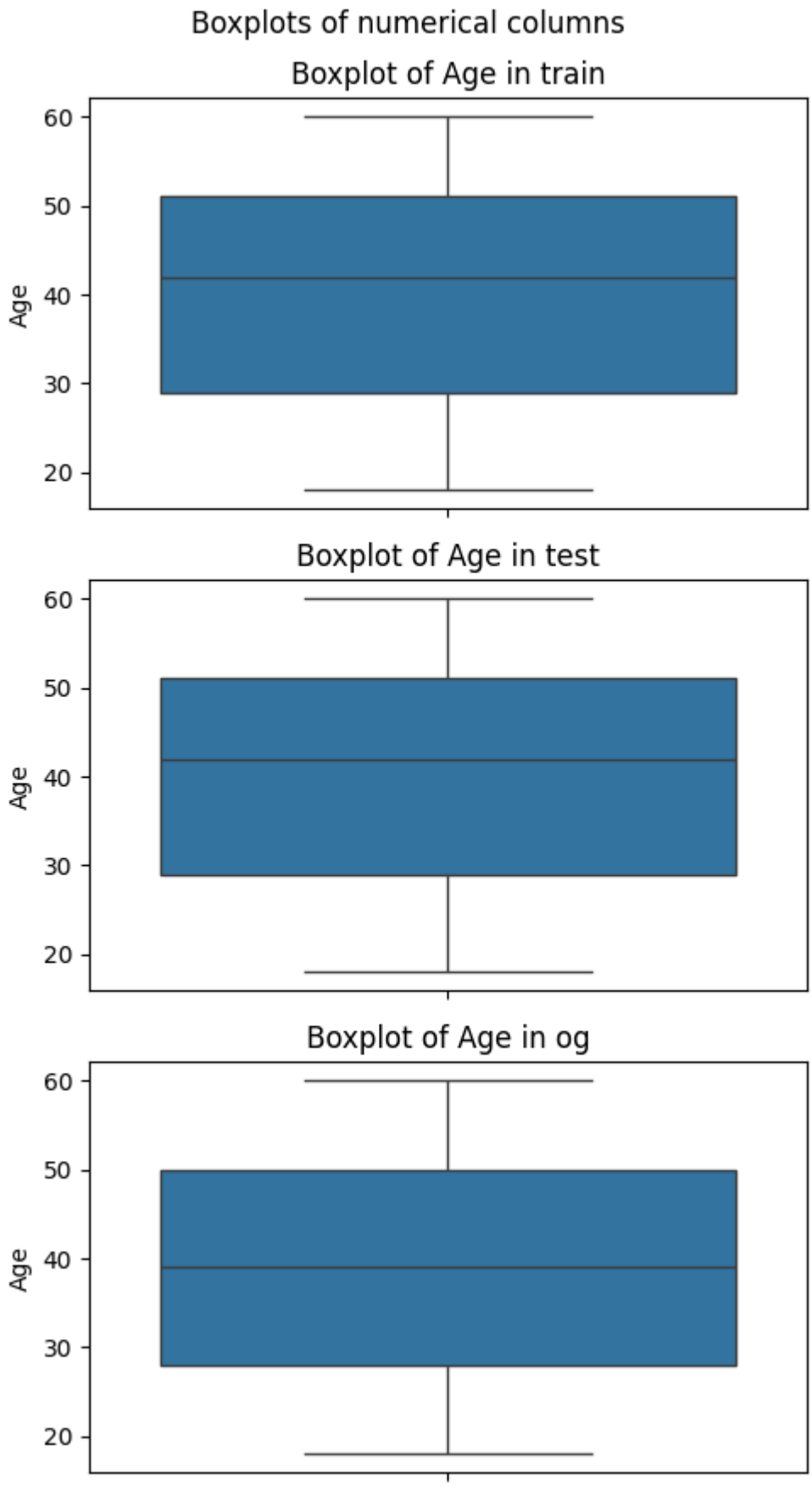
Out[211…

	Age
111479	56
31411	59
31582	26
138124	24
74553	57

In [212…

```
#Now we check for outliers using a boxplot
palettes = ['crest', 'Set2', 'magma']
fig, ax = plt.subplots(3, 1, figsize=(5,9))
fig.suptitle('Boxplots of numerical columns')
```

```
for i, dataset in enumerate(datasets_to_vizualize):
    for j, num_col in enumerate(num_cols):
        ax[i].set_title(f'Boxplot of {num_col} in {dataset}')
        df = eval(dataset)
        sns.boxplot(data=df, y=num_col, ax=ax[i])
plt.tight_layout()
plt.savefig('graphs/age_boxplots.jpg', dpi=300)
plt.show()
#We see that there are no particular outliers in the numerical columns
```



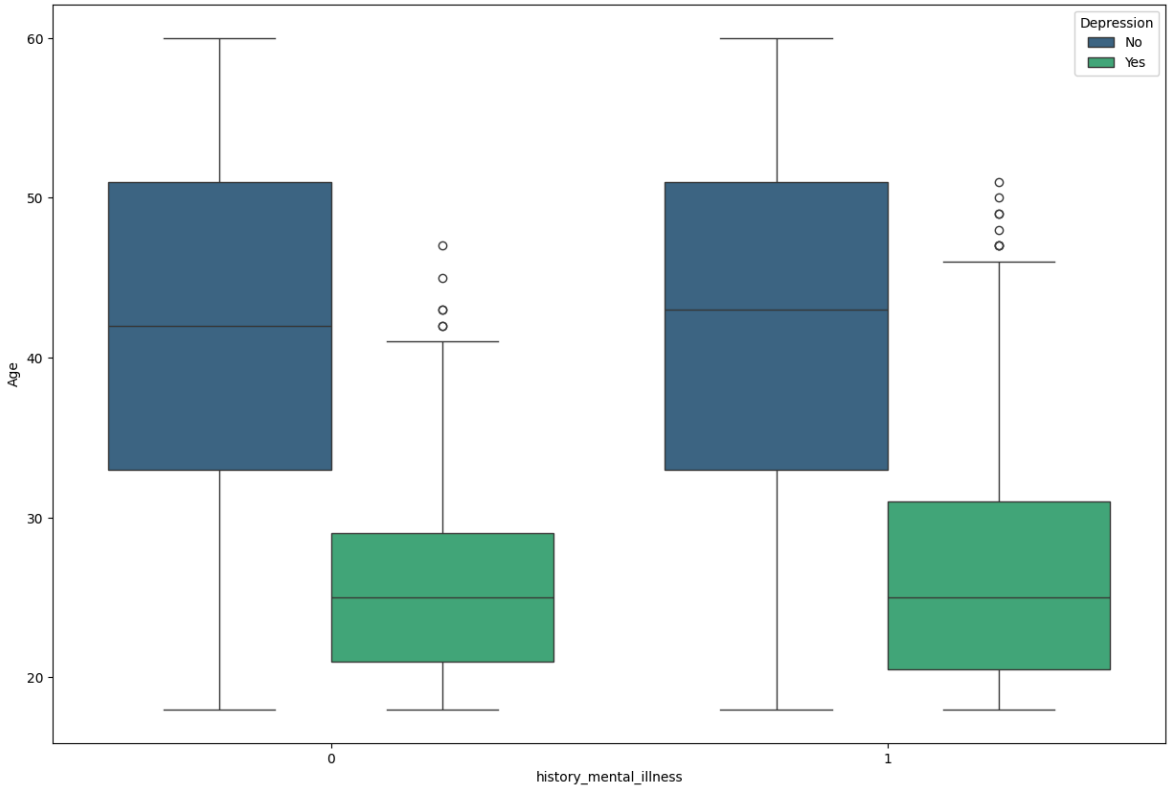
Deal with ouliers pls

In [213…

```
plt.figure(figsize=(15, 10))
sns.boxplot(data=og,x='history_mental_illness', y='Age', palette='viridis', hue=
```

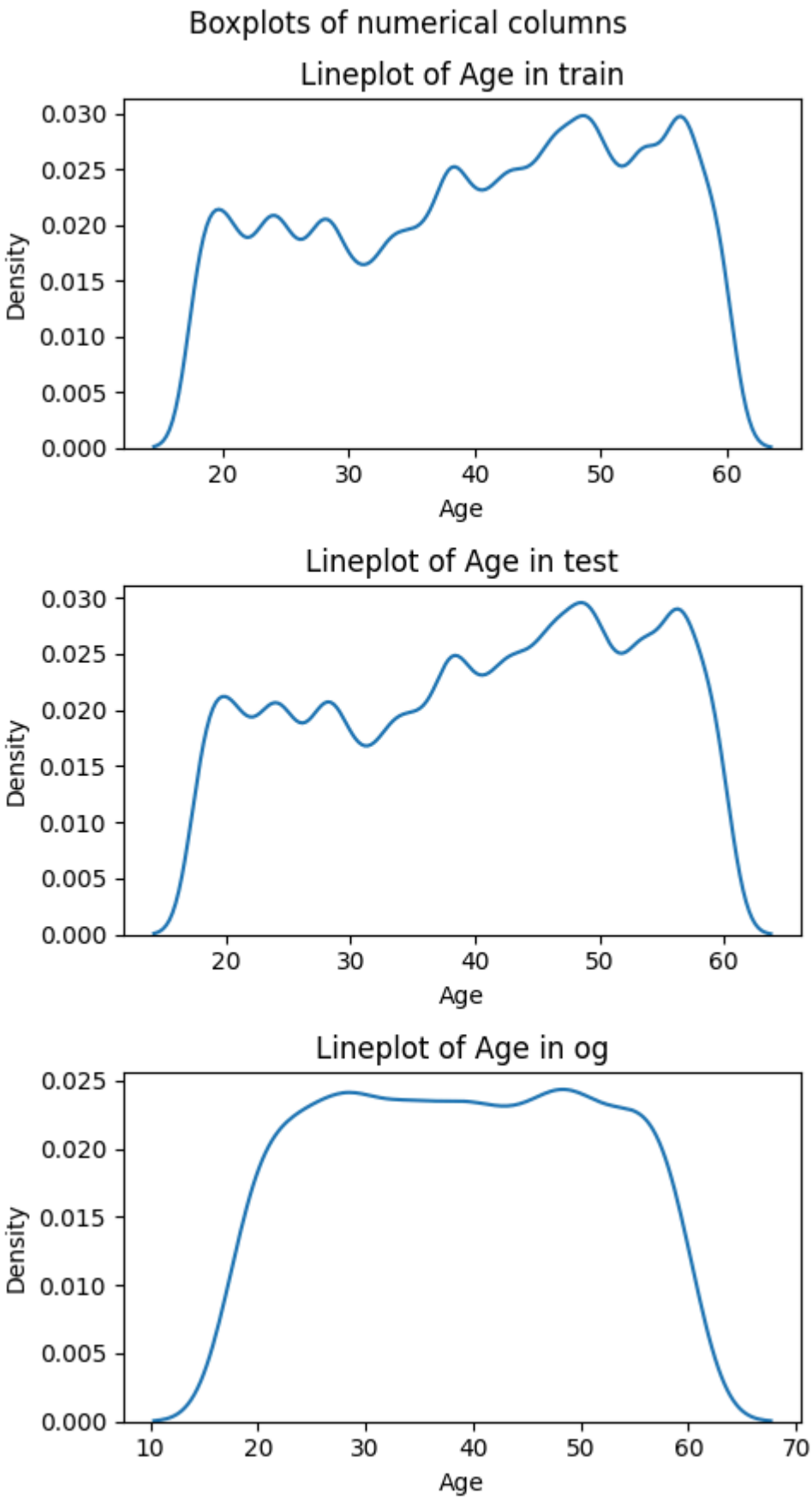
Out[213…

```
<Axes: xlabel='history_mental_illness', ylabel='Age'>
```



In [214…

```
# Checking for the distribution of the numerical columns
fig, ax = plt.subplots(3, 1, figsize=(5,9))
fig.suptitle('Boxplots of numerical columns')
for i, dataset in enumerate(datasets_to_vizualize):
    for j, num_col in enumerate(num_cols):
        ax[i].set_title(f'Lineplot of {num_col} in {dataset}')
        df = eval(dataset)
        sns.kdeplot(
            data=df,
            x='Age',
            ax=ax[i]
            )
plt.tight_layout()
plt.savefig('graphs/age_histplot_before.jpg', dpi=300)
plt.show()
```



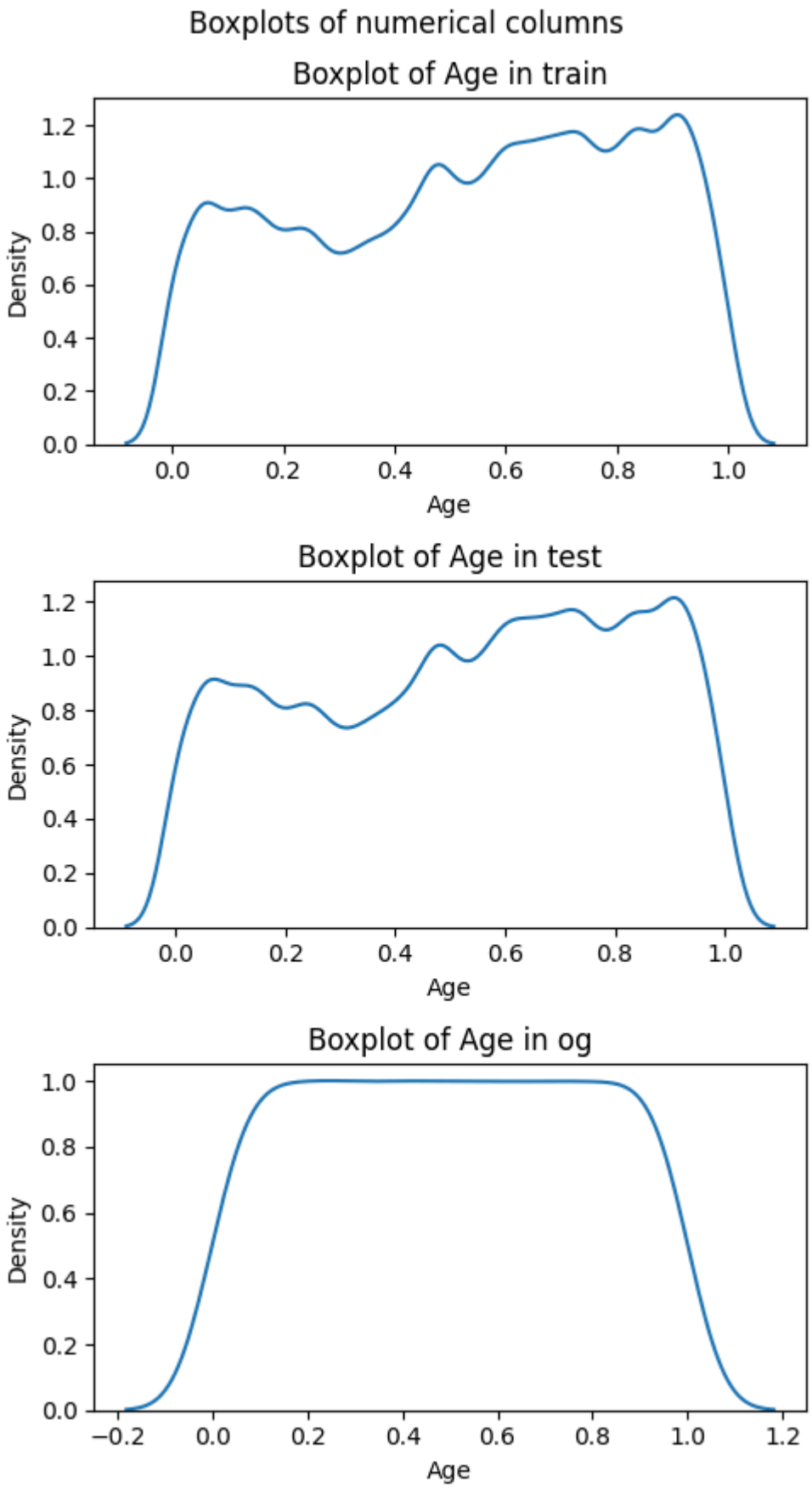
In [215…

```python
#Since the age column is not rightly uniformly distributed, we will use the Quan
from sklearn.preprocessing import QuantileTransformer

# Create a quantile transformer with uniform output
transformer = QuantileTransformer(output_distribution='uniform')

# Fit and transform the data
```

```
og['Age'] = transformer.fit_transform(og['Age'].to_numpy().reshape(-1, 1))
train['Age'] = transformer.transform(train['Age'].to_numpy().reshape(-1, 1))
test['Age'] = transformer.transform(test['Age'].to_numpy().reshape(-1, 1))
```

In [216…

```
# Checking for the distribution of the numerical columns
fig, ax = plt.subplots(3, 1, figsize=(5,9))
fig.suptitle('Boxplots of numerical columns')
for i, dataset in enumerate(datasets_to_vizualize):
    for j, num_col in enumerate(num_cols):
        ax[i].set_title(f'Boxplot of {num_col} in {dataset}')
        df = eval(dataset)
        sns.kdeplot(
            data=df,
            x='Age',
            ax=ax[i]
            )
plt.tight_layout()
plt.savefig('graphs/age_histplot_after.jpg', dpi=300)
plt.show()
```

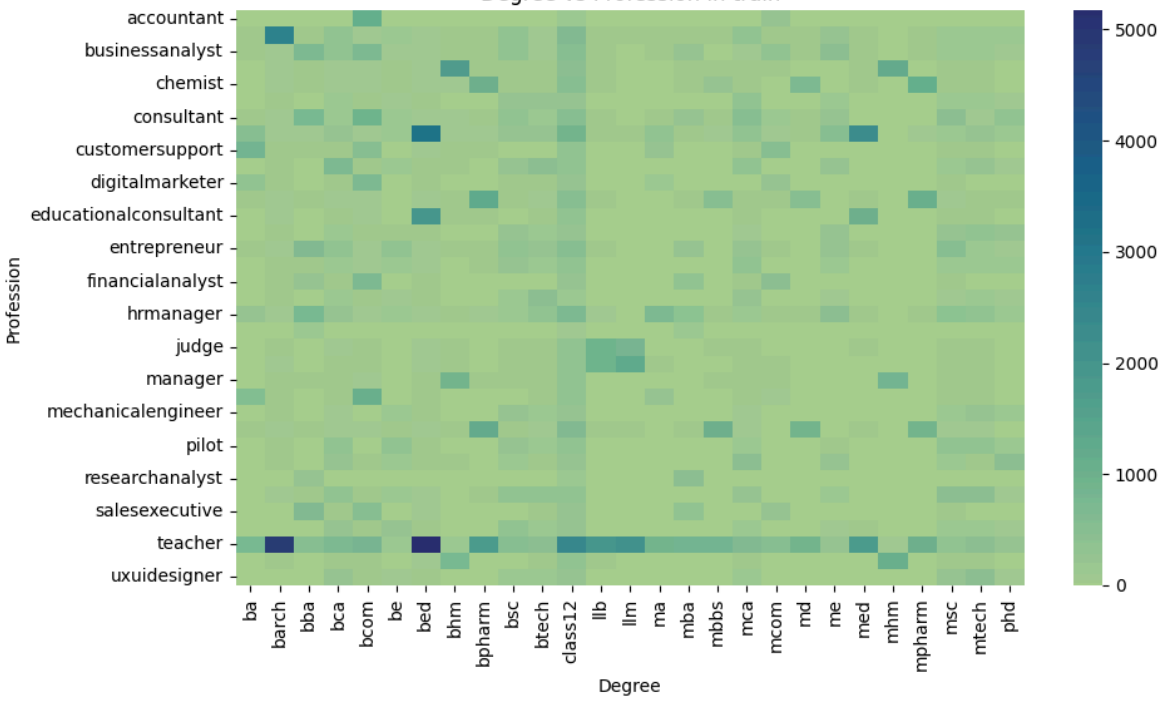


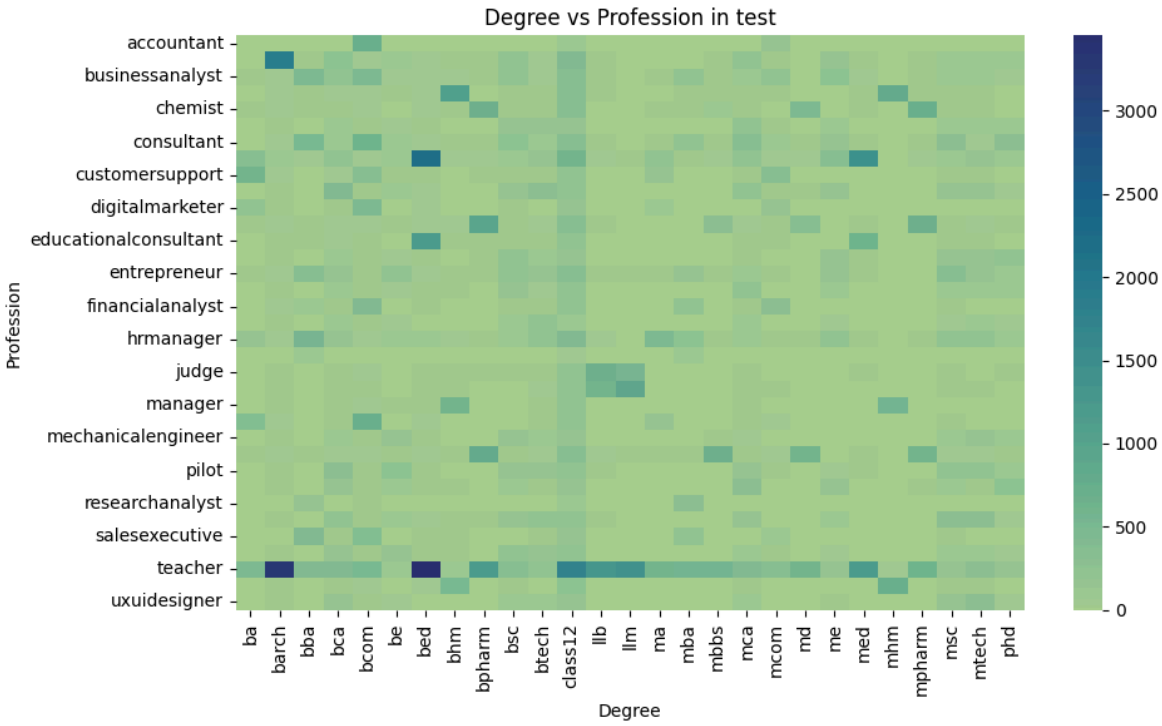
In [217…

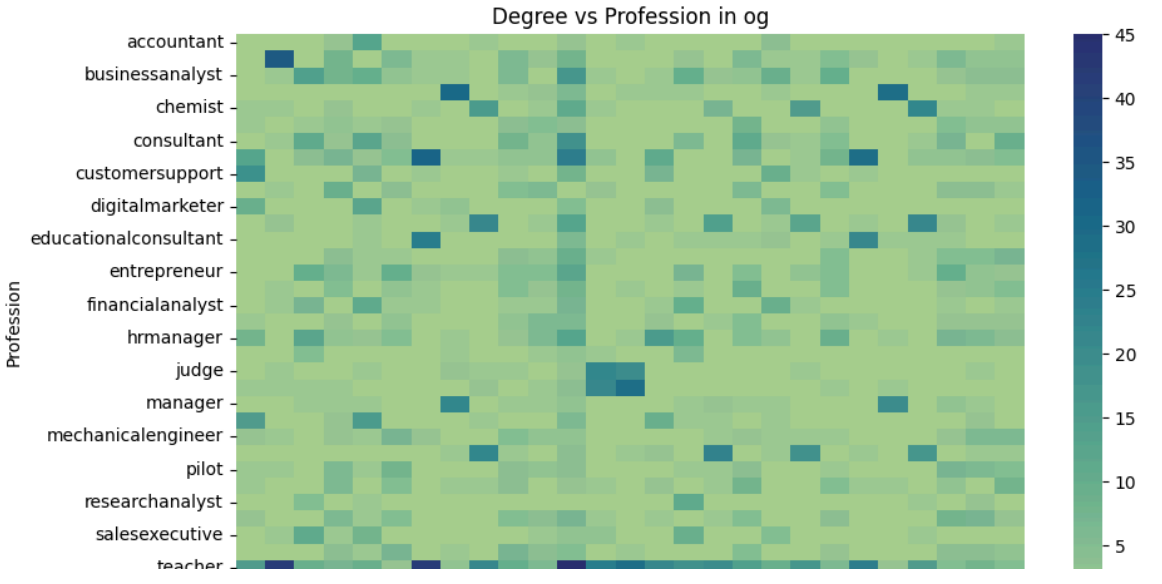
```
#Let's plot degree vs profession
fig, ax = plt.subplots(3,1, figsize=(10,18))
fig.suptitle('Degree vs Profession')
for i, dataset in enumerate(datasets_to_vizualize):
    ax[i].set_title(f'Degree vs Profession in {dataset}')
    df = eval(dataset)
    # sns.scatterplot(data=df,x='Profession', y='Degree', ax=ax[i], palette='cre
```

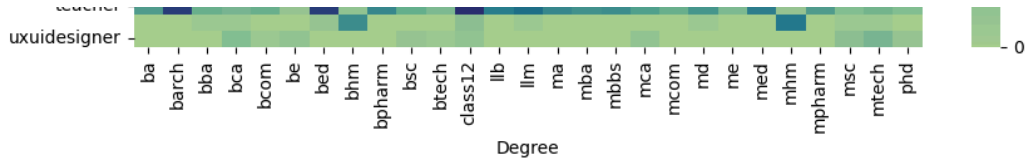
```
    temp_crosstab = pd.crosstab(df['Profession'], df['Degree'])
    sns.heatmap(temp_crosstab, annot=False, cmap='crest', ax=ax[i])
    ax[i].tick_params(axis='x', rotation=90)
plt.tight_layout()
# plt.savefig('graphs/degree_vs_profession.jpg', dpi=300)

#Conclusion : The train and test datasets agree on the distribution of the og da
```

Degree vs Profession









In [218…
```
train.columns
```

Out[218…
```
Index(['is_male', 'Age', 'City', 'is_student', 'Profession', 'Work Pressure',
       'Job Satisfaction', 'Sleep Duration', 'Dietary Habits', 'Degree',
       'Suicidal Thoughts', 'Work/Study Hours', 'Financial Stress',
       'history_mental_illness', 'Depression'],
      dtype='object')
```

In [219…
```
og.sample()
```

Out[219…

	is_male	Age	City	is_student	Profession	Work Pressure	Job Satisfaction	Du
2342	1	0.18969	Lucknow	1	softwareengineer	3.0	2.0	

Preprocessing the data

Divide the dataset into high and low cardinality cols

In [220…
```
columns_to_embed = ['Degree','City' ,'Profession']
columns_to_one_hot_encode = ['Dietary Habits', 'Sleep Duration']

combined_dataset = pd.concat([train, og], axis=0)

#Now, lets divide the dataset into embeddings and one hot encoded columns
embeddings_dataset = combined_dataset[columns_to_embed]
ohe_dataset = combined_dataset[columns_to_one_hot_encode]
test_ohe = test[columns_to_one_hot_encode]
y = combined_dataset['Depression']
combined_dataset.drop('Depression', axis=1, inplace=True)
```

In [221…
```
combined_dataset['Dietary Habits'].value_counts()
```

Out[221…
```
Dietary Habits
Moderate     50431
Unhealthy    47013
Healthy      45505
Name: count, dtype: int64
```

One hot encode low cardinality features

In [222…
```
#First, we need to decide which columns we have to preprocess
#We will one hot encode
#One hot encode the columns
ohe_dataset = pd.get_dummies(ohe_dataset, columns=columns_to_one_hot_encode, pre
test_ohe = pd.get_dummies(test_ohe, columns=columns_to_one_hot_encode, prefix='o
ohe_dataset.head()
```

Out[222...

	ohe_Moderate	ohe_Unhealthy	ohe_7-8 hours	ohe_Less than 5 hours	ohe_More than 8 hours
0	0	0	0	0	1
1	0	1	0	1	0
2	0	0	0	0	0
3	1	0	0	1	0
4	0	1	0	0	0

In [223...

```
combined_dataset.drop(columns=columns_to_one_hot_encode, inplace=True)
combined_dataset = pd.concat([combined_dataset, ohe_dataset], axis=1)
test.drop(columns=columns_to_one_hot_encode, inplace=True)
test = pd.concat([test, test_ohe], axis=1)
combined_dataset.sample(5)
```

Out[223...

	is_male	Age	City	is_student	Profession	Work Pressure	Satis
28867	1	0.925425	Ahmedabad	0	electrician	3.0	
124153	0	0.382382	Mumbai	1	digitalmarketer	5.0	
106004	0	0.829329	Vasai-Virar	0	marketingmanager	5.0	
39567	1	0.853353	Visakhapatnam	0	salesexecutive	4.0	
130166	1	0.000000	Pune	1	marketingmanager	2.0	

Embed the information cols

In [224...

```
#Before embedding, I need to deal with the information cols in the embeddings da
# I will use frequency encoding for the columns_to_embed
for col in columns_to_embed:
    combined_dataset[col] = combined_dataset[col].map(combined_dataset[col].valu

for col in columns_to_embed:
    test[col] = test[col].map(test[col].value_counts().to_dict())
combined_dataset.sample(5)
```

Out[224...

	is_male	Age	City	is_student	Profession	Work Pressure	Job Satisfaction	Degree
18298	1	0.264264	4947	1	4626	1.0	3.0	4422
7247	1	0.382382	4711	0	4626	5.0	4.0	8220
33740	1	0.925425	5303	0	5419	3.0	5.0	4620
18488	1	0.762262	4362	0	2418	3.0	4.0	8220
111631	0	0.968468	4647	0	2778	2.0	1.0	4390

Handling data imbalance

In [225...

```
from imblearn.over_sampling import SMOTE

# Ensure that y contains discrete class labels
y = y.apply(lambda x: 1 if x == 'Yes' else 0)
y = y.astype(int)

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(combined_dataset, y)
```

Build and train a model

In [226...

```
from sklearn.model_selection import train_test_split

# Split the data into features and target
X = X_resampled
y = y_resampled
#Now we concatenate the og dataset to X and y
# X = pd.concat([X, og.drop(['Depression','Degree', 'Profession','City'], axis=1
# y = pd.concat([y, og['Depression']])

#Now we will use StartifiedKFold to split the data
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

In [227...

```
#Now , I want to train a CatBoost model
from catboost import CatBoostClassifier
from sklearn.metrics import classification_report, accuracy_score

# Create a CatBoost model
model = CatBoostClassifier(iterations=200, learning_rate=0.1, depth=4, loss_func

# Fit the model
model.fit(X_train, y_train, eval_set=(X_test, y_test))
```

```
# Make predictions
train_preds = model.predict(X_train)
val_preds = model.predict(X_test)



#Now, I will make predictions on the test data
# test_preds = model.predict(test)
# test_preds
```

```
0:      learn: 0.8246159        test: 0.8254294 best: 0.8254294 (0)     total: 3
3.1ms   remaining: 6.58s
199:    learn: 0.9981359        test: 0.9985438 best: 0.9985438 (196)   total: 8.
19s     remaining: 0us

bestTest = 0.998543783
bestIteration = 196

Shrink model to first 197 iterations.
```

In [267…

```
# Print the classification report and accuracy
print('Training Classification Report:')
print(classification_report(y_train, train_preds, zero_division=0))
print('Validation Classification Report:')
print(classification_report(y_test, val_preds))
print('Training Accuracy:', accuracy_score(y_train, train_preds))
print('Validation Accuracy:', accuracy_score(y_test, val_preds))
```

```
Training Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    113996
           1       1.00      1.00      1.00    113995

    accuracy                           1.00    227991
   macro avg       1.00      1.00      1.00    227991
weighted avg       1.00      1.00      1.00    227991

Validation Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28498
           1       1.00      1.00      1.00     28499

    accuracy                           1.00     56997
   macro avg       1.00      1.00      1.00     56997
weighted avg       1.00      1.00      1.00     56997

Training Accuracy: 0.9981358913290437
Validation Accuracy: 0.9985437830061231
```

In [286…

```
final_df = model.predict(test)
# final_df = pd.Series(final_df, name='Depression')
# final_df = pd.concat([ids,final_df])
# final_df.to_csv('submission.csv', index=False)
final_df = pd.Series(final_df, name='Depression')
final_df = pd.concat([ids, final_df], axis=1)
final_df.to_csv('submission.csv', index=False)
```