**PUBLIC HEALTH AWARENESS**

**TEAM MEMBER**

REGISTER NUMBER: 412621243006

NAME: AMAL VIJAY G

# **Phase-3 Document Submission**

**Phase 3:Development Part 1:**

**Data Collection**

The data was collected from the Mental Health in Tech Survey dataset on Kaggle. This dataset contains data on mental health conditions, stress levels, and other related factors among tech workers.

**Data Retrieval**

The following relevant attributes were retrieved from the dataset:

* Age
* Gender
* Country
* State
* Job title
* Years of experience
* Self-employment status
* Family history of mental illness
* Treatment status
* Access to treatment resources
* How mental health affects work
* Number of employees at company
* Remote work status
* Tech company status
* Employer-provided mental health benefits
* Employer-provided wellness program
* Anonymity protection for seeking treatment
* Ease of medical leave for mental health conditions
* Consequences of discussing physical or medical matters with employers

**Data Cleaning**

The dataset was cleaned by:

* Handling missing data by imputation: Missing values were imputed using the mean or median value for the corresponding variable.
* Removing duplicates and irrelevant columns: Duplicate rows and irrelevant columns were removed from the dataset.
* Standardizing date formats and column names: Date formats and column names were standardized to ensure consistency.
* Correcting any inconsistencies or errors in the data: Any inconsistencies or errors in the data were corrected.

**Data Transformation**

The following new variables were created to represent the following:

* Age category (e.g., 18-25, 26-35, etc.)
* Mental health severity (e.g., mild, moderate, severe)

**Data Exploration**

Initial exploratory data analysis (EDA) was conducted to understand the characteristics of the dataset. Summary statistics, visualizations, and plots were created to gain insights into the data. For example, the following visualizations were created:

* Histogram of age
* Box plot of stress levels by gender
* Bar chart of the most common mental health conditions

**Data Validation**

The data was checked for anomalies, outliers, and inconsistencies. Mental health professionals also reviewed the data to ensure that it was accurate and reliable.

**Data Preprocessing**

The following preprocessing steps were performed:

* Normalization of age and stress level variables
* Encoding of categorical variables (e.g., gender, country, mental health condition)

**Data Splitting**

Data splitting into training and testing sets was not necessary for this project.

**Data Documentation**

A detailed report was created to document the data collection and preprocessing steps. This report includes information on the sources of the data, the cleaning and transformation steps performed, and the rationale for each step.

**Dataset Link:** <https://www.kaggle.com/datasets/osmi/mental-health-in-tech-survey>



**PYTHON PROGRAM:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import tensorflow as tf

data = pd.read\_csv('../input/mental-health-in-tech-survey/survey.csv')

data

data.isna().sum()

data.isna().mean()

data = data.drop('comments', axis=1)

data = data.drop('state', axis=1)

data['self\_employed'].unique()

data['self\_employed'].mode()

data['self\_employed'] = data['self\_employed'].fillna('No')

data['work\_interfere'].unique()

data['work\_interfere'].mode()

data['work\_interfere'] = data['work\_interfere'].fillna('Sometimes')

data

{column: len(data[column].unique()) for column in data.select\_dtypes('object').columns}

{column: list(data[column].unique()) for column in data.select\_dtypes('object').columns}

def encode\_gender(x):

if x.lower()[0] == 'f':

return 0

elif x.lower()[0] == 'm':

return 1

else:

return 2

data['Gender'] = data['Gender'].apply(encode\_gender)

target = 'treatment'

binary\_features = [

'self\_employed',

'family\_history',

'remote\_work',

'tech\_company',

'obs\_consequence'

]

ordinal\_features = [

'work\_interfere',

'no\_employees'

]

nominal\_features = [

'Country',

'benefits',

'care\_options',

'wellness\_program',

'seek\_help',

'anonymity',

'leave',

'mental\_health\_consequence',

'phys\_health\_consequence',

'coworkers',

'supervisor',

'mental\_health\_interview',

'phys\_health\_interview',

'mental\_vs\_physical'

]

def binary\_encode(df, columns, positive\_values):

df = df.copy()

for column, positive\_value in zip(columns, positive\_values):

df[column] = df[column].apply(lambda x: 1 if x == positive\_value else 0)

return df

def ordinal\_encode(df, columns, orderings):

df = df.copy()

for column, ordering in zip(columns, orderings):

df[column] = df[column].apply(lambda x: ordering.index(x))

return df

def onehot\_encode(df, columns, prefixes):

df = df.copy()

for column, prefix in zip(columns, prefixes):

dummies = pd.get\_dummies(df[column], prefix)

df = pd.concat([df, dummies], axis=1)

df = df.drop(column, axis=1)

return df

binary\_positive\_values = ['Yes' for feature in binary\_features]

ordinal\_orderings = [

['Never', 'Rarely', 'Sometimes', 'Often'],

['1-5', '6-25', '26-100', '100-500', '500-1000', 'More than 1000']

]

nominal\_prefixes = [

'co',

're',

'be',

'ca',

'we',

'se',

'an',

'le',

'mc',

'ph',

'cw',

'su',

'mi',

'pi',

'mp'

]

data = binary\_encode(

data,

columns=binary\_features,

positive\_values=binary\_positive\_values

)

data = ordinal\_encode(

data,

columns=ordinal\_features,

orderings=ordinal\_orderings

)

data = onehot\_encode(

data,

columns=nominal\_features,

prefixes=nominal\_prefixes

)

data

data = binary\_encode(data, columns=['treatment'], positive\_values=['Yes'])

data

print("Remaining non-numeric columns:", len(data.select\_dtypes('object').columns))

{column: len(data[column].unique()) for column in data.select\_dtypes('object').columns}

data = data.drop('Timestamp', axis=1)

print("Remaining non-numeric columns:", len(data.select\_dtypes('object').columns))

print("Remaining missing values:", data.isna().sum().sum())

y = data['treatment'].copy()

X = data.drop('treatment', axis=1).copy()

scaler = StandardScaler()

X = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, random\_state=100)

X.shape

print("Class Distribution (Positive to Negative): {:.1f}% / {:.1f}%".format(y\_train.mean() \* 100, (1 - y\_train.mean()) \* 100))

inputs = tf.keras.Input(shape=(X.shape[1],))

x = tf.keras.layers.Dense(1024, activation='relu')(inputs)

x = tf.keras.layers.Dense(1024, activation='relu')(x)

outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

model = tf.keras.Model(inputs, outputs)

model.compile(

optimizer='adam',

loss='binary\_crossentropy',

metrics=[

'accuracy',

tf.keras.metrics.AUC(name='auc')

]

)

batch\_size = 64

epochs = 50

history = model.fit(

X\_train,

y\_train,

validation\_split=0.2,

batch\_size=batch\_size,

epochs=epochs,

callbacks=[

tf.keras.callbacks.ReduceLROnPlateau()

]

)

plt.figure(figsize=(12, 6))

plt.plot(range(epochs), history.history['accuracy'], label="Training Accuracy")

plt.plot(range(epochs), history.history['val\_accuracy'], label="Validation Accuracy")

plt.xlabel("Epoch")

plt.ylabel("Accuracy")

plt.legend()

plt.title("Accuracy Over Time")

plt.show()

model.evaluate(X\_test, y\_test)

**OUTPUT:**

