7.0 Appendix 1

Importation of libraries and dataset/dataset display/Descriptive statistics

```
import pandas as pd

import matplotlib.pyplot as plt

import numpy as npimport seaborn as sns

import matplotlib.pyplot as plt

from math import pi

import string

import re

import researchpy as rp

from nltk.corpus import stopwords

from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import roc_curve, roc_auc_score

from sklearn.preprocessing import LabelEncoder

from scipy.stats import chi2_contingency

from matplotlib.colors import Normalize

from sklearn.model_selection import train_test_split
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score
import statsmodels.api as sm
import squarify
import math
```

```
# Read the Excel file into a DataFrame

data = pd.read_excel('C:/Users/Administrator/Desktop/bradford project/group assignment/GROUP_ASSIGNMENT_DATA.xlsx')


# display the dataframe

data.head()
```

REMOVING MISSING VALUES AND PLOTTING VALUES VARIABLES.

```
 # Remove rows where st_slope is 0

data.drop(data[data.Age == 0].index, inplace=True)


# Group age into different categories

age_groups = []

for i in range(0, len(data)):

    if (data['Age'].iloc[i] > 0) and (data['Age'].iloc[i] < 20):

        age_groups.append('0-20')

    elif (data['Age'].iloc[i] > 20) and (data['Age'].iloc[i] < 40):

        age_groups.append('21-40')

    elif (data['Age'].iloc[i] > 40) and (data['Age'].iloc[i] < 60):
```

```
        age_groups.append('41-60')
    else:
        age_groups.append('>60')
data['Age_Group'] = age_groups

# Plot the distribution of age groups
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 5))
data['Age_Group'].value_counts().plot(kind='bar', ax=ax1)
plt.title('Age Group Distribution')

data['Job_Type_Interest'] = pd.to_numeric(data['Job_Type_Interest'], errors='coerce')

# Plot the job type interest by age group and gender
sns.barplot(x='Age_Group', y='Job_Type_Interest', hue='Gender', data=data, palette="rocket", ci=None, ax=ax2)
plt.title('Job Type Interest by Age Group and Gender')

# Group salary expectations into different categories
salary_groups = []
```

```
for i in range(0, len(data)):

    if data['Salary_Expectation_(in_USD)'].iloc[i] < 20000:

        salary_groups.append('Less than $20,000')

    elif data['Salary_Expectation_(in_USD)'].iloc[i] < 40000:

        salary_groups.append('$20,000 - $40,000')

    elif data['Salary_Expectation_(in_USD)'].iloc[i] < 60000:

        salary_groups.append('$40,000 - $60,000')

    else:

        salary_groups.append('More than $60,000')

data['Salary_Group'] = salary_groups



# Plot the distribution of salary groups

plt.figure(figsize=(18, 12))

plt.subplot(233)

sns.countplot(x='Salary_Group', data=data, palette='Set2')

plt.title('Salary Expectation Distribution')
```

DATA CLEANING AND PRE-PROCESSING:

Replace values in the desired_company column

data['desired_company'][data['desired_company'] == 'Google'] = 'Technology'

data['desired_company'][data['desired_company'] == 'Microsoft'] = 'Technology'

data['desired_company'][data['desired_company'] == 'Goldman Sachs'] = 'Finance'

data['desired_company'][data['desired_company'] == 'JPMorgan Chase'] = 'Finance'

data['desired_company'][data['desired_company'] == 'Procter & Gamble'] = 'Consumer Goods'

data['desired_company'][data['desired_company'] == 'Unilever'] = 'Consumer Goods'

data['desired_company'][data['desired_company'] == 'McKinsey & Company'] = 'Consulting'

data['desired_company'][data['desired_company'] == 'Boston Consulting Group'] = 'Consulting'

EXPLORATORY DATA ANALYSIS:

Continuous Features Distribution

plt.style.use("dark_background")

plt.figure(figsize=(18,12))

plt.subplot(321)

```
diag= sns.distplot(data['Age'], rug=True, color='yellow',label='Skewness: %.2f'%data['Age'].skew())

plt.title("Age Distribution",fontsize=20,fontweight="bold")

plt.grid(False)

plt.legend()


# Data plot with statistics

d1=data[data['Job_Type_Interest']=='Full-time']

d2=data[data['Job_Type_Interest']=='Part-time']


# Plotting correlation heatmap

plt.style.use('seaborn')

plt.figure(figsize=(15,8))

sns.heatmap(data[['Age', 'Salary_Expectation_(in_USD)']].corr(), annot=True, cmap='Blues')

plt.title('Correlation between Age and Salary Expectation (in USD)')


# Histogram

plt.figure(figsize=(8,6))
```

```
sns.histplot(data=data, x='Salary_Expectation_(in_USD)', bins=10, kde=True)

plt.title('salary distribution', fontsize=16)

plt.xlabel('Salary_Expectation_(in_USD)', fontsize=14)

plt.ylabel('Count', fontsize=14)

plt.show()


# Select the numerical feature(s) to plot

numerical_feature = "Age"


# Plot the distribution using Seaborn

sns.displot(data[numerical_feature], kde=True)


# Set the plot title and axis labels

plt.title(f"Distribution of {numerical_feature}")

plt.xlabel(numerical_feature)

plt.ylabel("Count")


# Show the plot
```

```
plt.show()
```

HYPOTHESIS

```
# Define hypothes test whether the mean age of the sample is greater than 30

null_hypothesis = "The population mean age is <= 30"

alternative_hypothesis = "The population mean age is > 30"


# Perform one-sample t-test

t_statistic, p_value = st.ttest_1samp(data['Age'], 30)


# Generate plot of the age distribution

plt.hist(data['Age'], density=True, alpha=0.5)

plt.axvline(data['Age'].mean(), color='red', linestyle='dashed', linewidth=1)

plt.title("Age Distribution of Sample")

plt.xlabel("Age")

plt.ylabel("Density")


# Set x-axis tick labels to show age values in integer format

plt.xticks(range(int(data['Age'].min()), int(data['Age'].max())+1, 5))
```

```
# Add t-test results to plot

plt.text(40, 0.25, "Null hypothesis: " + null_hypothesis)

plt.text(40, 0.23, "Alternative hypothesis: " + alternative_hypothesis)

plt.text(40, 0.21, "Sample mean: {:.2f}".format(data['Age'].mean()))

plt.text(40, 0.19, "Test statistic: {:.2f}".format(t_statistic))

plt.text(40, 0.17, "P-value: {:.3f}".format(p_value))

if p_value < 0.05:

   plt.text(40, 0.15, "The null hypothesis can be rejected at the 5% significance level")

else:

   plt.text(40, 0.15, "The null hypothesis cannot be rejected at the 5% significance level")


# Show plot

plt.show()


# Select the features to use in the heatmap

features = ['Age', 'Gender', 'Academic Background', 'Field of Study', 'Skills', 'Industry_Interest', 'Job_Type_Interest', 'Location_Interest', 'Salary_Expectation_(in_USD)']
```

```
# Compute the correlation matrix

corr = data[features].corr()


# Create a heatmap

sns.heatmap(corr, cmap="YlGnBu", annot=True)


# Show the plot

plt.show()
```

#NUMERICAL FEATURE DISTRIBUTION

```
# Define custom color palette

custom_palette = ['#4C72B0', '#C44E52']


# Specify numerical columns

numerical_cols = ['Age', 'Salary Expectation (in USD)']


# Plot numerical feature distribution
```

```
sns.set_style('whitegrid')

sns.pairplot(data[numerical_cols], kind='scatter', palette=custom_palette, height=2.5)

plt.show()
```

CHI-TEST

```
# Recode the gender column to be numeric (0 = F, 1 = M)

data['Gender_Numeric'] = data['Gender'].apply(lambda x: 0 if x == 'F' else 1)


# Create a contingency table of gender and field of study, and perform a chi-square test

crosstab, test, expected = rp.crosstab(data['Gender'], data['Field of Study'], test='chi-square', expected_freqs=True)


# Print the results of the chi-square analysis

print('Crosstab:')

print(crosstab)


print('\nChi-square test:')

print(test)
```

```
print('\nExpected frequencies:')

print(expected)


# Map each location to a region

region_map = {

    "New York": "East Coast",

    "Boston": "East Coast",

    "Washington D.C.": "East Coast",

    "Los Angeles": "West Coast",

    "San Francisco": "West Coast",

    "Seattle": "West Coast",

    "Chicago": "Midwest",

    "Detroit": "Midwest",

    "Minneapolis": "Midwest"

}

data["Region"] = data["Location_Interest"].map(region_map)


# Plot the distribution of salary expectation for each region
```

```
sns.boxplot(x="Region", y="Salary_Expectation_(in_USD)", data=data)

plt.xlabel("Region")

plt.ylabel("Salary Expectation (in USD)")

plt.title("Distribution of Salary Expectation by Region")

plt.show()
```

PREDICTION MODEL

```
data['Academic Background'] = pd.to_numeric(data['Academic Background'], errors='coerce')

data['Industry_Interest'] = pd.to_numeric(data['Industry_Interest'], errors='coerce')

data['Skills'] = pd.to_numeric(data['Skills'], errors='coerce')

data['Job_Type_Interest'] = pd.to_numeric(data['Job_Type_Interest'], errors='coerce')

data['Location_Interest'] = pd.to_numeric(data['Location_Interest'], errors='coerce')

data['Field of Study'] = pd.to_numeric(data['Field of Study'], errors='coerce')

data['desired_company'] = pd.to_numeric(data['desired_company'], errors='coerce')

data['Name'] = pd.to_numeric(data['Name'], errors='coerce')

data['Gender'] = pd.to_numeric(data['Gender'], errors='coerce')
```

```
# Separate the target variable (desired company) from the features
y = data['desired_company']


# Create a missing indicator for X and y
X_indicator = MissingIndicator()
y_indicator = MissingIndicator()

X_missing = X_indicator.fit_transform(X)
y_missing = y_indicator.fit_transform(y.values.reshape(-1, 1))

# Impute missing values in X using the median strategy
imputer = SimpleImputer(strategy='median')
X = imputer.fit_transform(X)

# Impute missing values in y using the most frequent strategy
imputer = SimpleImputer(strategy='most_frequent')
y = imputer.fit_transform(y.values.reshape(-1,1)).ravel()
```

```
# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



# Train a random forest classifier on the training set

clf = RandomForestClassifier()

clf.fit(X_train, y_train)



# Make predictions on the test data and calculate accuracy score

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)



print("Accuracy:", accuracy)
```

MODEL PREDICTION

#FEATURE ENGINEERING

```
# Filter out future warnings from scikit-learn

warnings.filterwarnings("ignore", category=FutureWarning)
```

```python
# Fill missing values if any
data.fillna(value={'Salary Expectation (in USD)': 0}, inplace=True)

# Feature Engineering

# Strip whitespace from column names
data.columns = data.columns.str.strip()

# Check if 'Title' column is present
if 'Title' in data.columns:
    # Feature: JobTitle_Length (length of job title)
    data['JobTitle_Length'] = data['Title'].apply(lambda x: len(str(x)))

# Check if 'Skills' column is present
if 'Skills' in data.columns:
    # Feature: Skills_Count (number of skills)
    data['Skills_Count'] = data['Skills'].apply(lambda x: len(str(x).split(',')))
```

```
# Check if 'Academic Background' column is present
if 'Academic Background' in data.columns:
    # Feature: AcademicBackground_Encoded (label encoding for academic background)
    label_encoder = LabelEncoder()
    data['AcademicBackground_Encoded'] = label_encoder.fit_transform(data['Academic Background'].astype(str))

# Check if 'Field of Study' column is present
if 'Field of Study' in data.columns:
    # Feature: FieldOfStudy_Tfidf (TF-IDF vectorization for field of study)
    tfidf_vectorizer = TfidfVectorizer()
    field_of_study_tfidf = tfidf_vectorizer.fit_transform(data['Field of Study'].fillna('').astype(str))
    field_of_study_tfidf_df = pd.DataFrame(field_of_study_tfidf.toarray(), columns=tfidf_vectorizer.get_feature_names_out())
    data = pd.concat([data, field_of_study_tfidf_df], axis=1)

# Check if 'Industry Interest' column is present
```

```
if 'Industry Interest' in data.columns:
    # Feature: IndustryInterest_OHE (one-hot encoding for industry interest)
    one_hot_encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
    industry_interest_ohe = one_hot_encoder.fit_transform(data['Industry Interest'].fillna('').astype(str).values.reshape(-1, 1))
    industry_interest_ohe_df = pd.DataFrame(industry_interest_ohe, columns=one_hot_encoder.get_feature_names_out())
    data = pd.concat([data, industry_interest_ohe_df], axis=1)

# Check if 'Salary Expectation (in USD)' column is present
if 'Salary Expectation (in USD)' in data.columns:
    # Feature: Salary_Scaled (scaling salary expectation using standard scaler)
    scaler = StandardScaler()
    data['Salary_Scaled'] = scaler.fit_transform(data['Salary Expectation (in USD)'].values.reshape(-1, 1))

# Drop original columns used for feature engineering
columns_to_drop = [col for col in ['Title', 'Skills', 'Academic Background', 'Field of Study', 'Industry Interest', 'Salary Expectation (in USD)'] if col in data.columns]
```

data.drop(columns_to_drop, axis=1, inplace=True)

Save the updated dataset to Excel

data.to_excel('updated_dataset.xlsx', index=False)

Test/Training and recommendation

Encode categorical variables

encoder = LabelEncoder()

data['Gender'] = encoder.fit_transform(data['Gender'])

Select the categorical columns for one-hot encoding

categorical_cols = ['Academic Background', 'Field of Study', 'Skills', 'Industry Interest', 'Job Type Interest', 'Location Interest']

Perform one-hot encoding using get_dummies

```
data_encoded = pd.get_dummies(data, columns=categorical_cols)


# Drop rows with missing values

data_encoded = data_encoded.dropna()


# Split the dataset into training and testing sets

train_data, test_data = train_test_split(data_encoded, test_size=0.2, random_state=42)


# Prepare the data for training

X_train = train_data.drop(['desired company', 'Name'], axis=1)

y_train = train_data['desired company']


X_test = test_data.drop(['desired company', 'Name'], axis=1)

y_test = test_data['desired company']


# Train the logistic regression model

model = LogisticRegression()

model.fit(X_train, y_train)
```

```
# Predict the probabilities of job interests for test data

y_proba = model.predict_proba(X_test)


# Convert the probabilities into a dataframe for easier analysis

proba_df = pd.DataFrame(y_proba, columns=model.classes_)


# Print the top recommended job for each student

for i, row in proba_df.iterrows():

    top_job = row.idxmax()

    print(f"Recommended job for student {i}: {top_job}")


# Evaluate the accuracy of the model

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

KNN ALGORITHMS

```
# Convert non-float columns to float

float_columns = ['Age', 'Salary Expectation (in USD)']

data[float_columns] = data[float_columns].astype(float)


# Convert non-numeric columns to numeric

data['Minimum Qualifications'] = data['Minimum Qualifications'].astype('category').cat.codes

data['Academic Background'] = data['Academic Background'].astype('category').cat.codes

data['Field of Study'] = data['Field of Study'].astype('category').cat.codes

data['Skills'] = data['Skills'].astype('category').cat.codes

data['Industry Interest'] = data['Industry Interest'].astype('category').cat.codes

data['Job Type Interest'] = data['Job Type Interest'].astype('category').cat.codes

data['Location Interest'] = data['Location Interest'].astype('category').cat.codes

data['desired company'] = data['desired company'].astype('category').cat.codes

data['Gender'] = data['Gender'].astype('category').cat.codes


# Split the data into features (X) and target (y)

X = data.drop(['user ID', 'Name', 'desired company','Title'], axis=1)  # Features
```

```
y = data['desired company']  # Target


# Split the data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=40)


# Create a K-Nearest Neighbors (KNN) classifier

knn = KNeighborsClassifier(n_neighbors=3)


# Train the classifier

knn.fit(X_train, y_train)


# Predict on the test set

y_pred = knn.predict(X_test)


# Calculate the accuracy of the model

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)
```

#NATURAL LANGUAGE

```
# Preprocess the data

data['Skills'] = data['Skills'].fillna('')  # Replace missing values with an empty string

data['Text'] = data['Academic Background'] + ' ' + data['Field of Study'] + ' ' + data['Skills']
+ ' ' + data['Industry Interest']


# Convert the text data into TF-IDF vectors

vectorizer = TfidfVectorizer()

tfidf_matrix = vectorizer.fit_transform(data['Text'].fillna(''))


# Compute the pairwise cosine similarity scores

cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)


# Function to recommend jobs based on user input

def recommend_jobs(user_skills, user_industry_interest):

   # Create a new DataFrame to store job recommendations

   recommendations = pd.DataFrame(columns=['desired company', 'Similarity Score'])


   # Combine user input into a single text

   user_text = ' '.join([user_skills, user_industry_interest])
```

```
    # Convert user input into a TF-IDF vector
    user_tfidf = vectorizer.transform([user_text])

    # Compute similarity scores between user input and job descriptions
    similarity_scores = cosine_similarity(user_tfidf, tfidf_matrix)

    # Get indices of top similarity scores
    top_indices = similarity_scores.argsort()[0][::-1]

    # Generate recommendations
    for idx in top_indices:
        job_title = data.loc[idx, 'desired company']
        similarity_score = similarity_scores[0][idx]
        recommendations = pd.concat([recommendations, pd.DataFrame({'desired company': [job_title], 'Similarity Score': [similarity_score]})], ignore_index=True)

    return recommendations
```

```
# Example usage
user_skills = 'programming data analysis machine learning'
user_industry_interest = 'technology'

recommendations = recommend_jobs(user_skills, user_industry_interest)
print(recommendations.head())


# Create empty lists to store accuracies for each k value
train_accuracies = []
test_accuracies = []
overall_accuracies = []

# Test k-NN for k=1 through 3
for k in range(1, 4):
    # Create the k-NN classifier with k=k
    knn = KNeighborsClassifier(n_neighbors=k)
```

```
# Train the model on the training set

knn.fit(X_train, y_train)


# Make predictions on the training set

y_train_pred = knn.predict(X_train)


# Calculate the accuracy on the training set

train_accuracy = accuracy_score(y_train, y_train_pred)

train_accuracies.append(train_accuracy)


# Make predictions on the test set

y_test_pred = knn.predict(X_test)


# Calculate the accuracy on the test set

test_accuracy = accuracy_score(y_test, y_test_pred)

test_accuracies.append(test_accuracy)


# Calculate the overall accuracy
```

```
    overall_accuracy = (train_accuracy + test_accuracy) / 2
    overall_accuracies.append(overall_accuracy)

# Plot the results
k_values = range(1, 4)
plt.plot(k_values, train_accuracies, label="Train Accuracy")
plt.plot(k_values, test_accuracies, label="Test Accuracy")
plt.plot(k_values, overall_accuracies, label="Overall Accuracy")
plt.xlabel("k")
plt.ylabel("Accuracy")
plt.title("Accuracy vs. k")
plt.legend()
plt.show()
```

8.0 Appendix 2

Dataset link

https://www.kaggle.com/datasets/ihetuemmanuel/job-recommendation-system

9.0 Appendix 3

Job recommendation system link

https://sonu08-job-recommendation-system-app-7z69ep.streamlit.app/

10.0 Appendix 4

Sn	Contributions	Names
1.	Chapter one	Funmilola Patience Usman
2.	Chapter two	Ganiu Sulaimon
3.	Chapter three	Adebisi Beatrice Olumoko
4.	Chapter four	Uju Judith Eziokwu
5.	Chapter five	Habeeb Oluwarotimi Oyekunle