Applied Stats - Data Analysis Project

Team 28 - Devesh Gautam, Ansh Bhatia, Sivanessan

Importing essential libraries for data analysis and visualisation

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [ ]:
```
# color palette for the plots (similar to pptx color palette)
pallete1 = ['#0f4662','#7994a0','#a9becb','#dbe5ea']
```

Reading the csv file

In [ ]:
```
data = pd.read_csv('data1.csv')
data.head()
```

In [ ]:
```
data.describe()
```

Converting non-integer data points to equvivalent integer values for the attribute - "Type of music"

1-Very Sad & Dark

2-Melancholic & Emotional

3-Neutral & Balanced

4-Upbeat & Cheerful

5-Extremely Happy & Euphoric

Plotting essential charts for different attributes

In [ ]:
```
# Histogram for CGPA distribution
plt.figure(figsize=(10,5))
sns.histplot(data['CGPA'], kde=False, color = pallete1[1],bins = 15)
plt.title('CGPA Distribution')
plt.ylabel('Number of Students', fontsize=12)
plt.xlabel('CGPA', fontsize=12)
plt.show()
```

In [ ]:
```
# Bar plot for attribute vs frequency for the whole numerical dataset
pallete1 = ['#0f4662','#7994a0','#a9becb','#dbe5ea']
for column in data.columns:
    if column not in ['Timestamp','CGPA','UG/PG','Year','musicHappy','musicHappy
        attribute = data[column].value_counts().sort_index()
        plt.figure(figsize=(10, 5))
        sns.barplot(x=attribute.index, y=attribute.values, alpha=0.8, color='#79

        # Set y-axis scale to integer number of students
```

```
        y_ticks = range(0, int(attribute.max() + 2), 2)
        plt.yticks(y_ticks)

        plt.ylabel('Number of Students', fontsize=12)
        plt.xlabel(column, fontsize=12)
        plt.title(f'{column} vs Frequency')
        plt.show()

        attribute_dist = data.groupby(column).size()
        print(attribute_dist)
```

In [ ]:

```
# pie chart for UG/PG students distribution
graduation_types = data['UG/PG'].value_counts()
plt.figure(figsize=(5,5))
explode = [0.1 if i == 0 else 0 for i in range(len(graduation_types))]
colors = ['#ffcc99','#FF0000','#99ff99','#ff9999','#ffcc99','#ffb3e6','#c4e17f',
plt.pie(graduation_types, autopct='%1.1f%%', colors=colors,startangle=140,shadow

stud_dist = data.groupby('UG/PG').size()
print(stud_dist)
```

In [ ]:

```
# Pie chart for type of songs people listen to
song_types = data['musicHappy'].value_counts()
colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99','#c2c2f0','#ffb3e6','#c4e17f',

plt.figure(figsize=(8, 8))
# plt.pie(song_types, labels=song_types.index, autopct='%1.1f%%', startangle=90,
plt.pie(song_types, autopct='%1.1f%%',explode=[0.1,0.1,0.1,0.2,0.1], startangle=
plt.title('Type of Songs People Listen To')
plt.legend(
    labels=song_types.index,
    title='Song Types',
    loc='center left',
    bbox_to_anchor=(1, 0, 0.5, 1)
)

plt.show()

song_dist = data.groupby('musicHappy').size()
print(song_dist)
```

In [ ]:

```
data['musicHappyInt'] = data['musicHappy'].str.extract(r'(\d+)').astype(int)
data.head()
```

Box plots for descriptive statistics analysis of the attributes CGPA, No of clubs, Difficulty of branch etc.

In [ ]:

```
need_boxPlot=['CGPA','clubsNo','branchDifficulty','religion','musicHappyInt','ha

for column in need_boxPlot:
    plt.figure(figsize=(10, 5))
    sns.boxplot(x=data[column], color=pallete1[1], orient='h', whis=(0, 100))
    plt.title(f'Box Plot for {column}')
    plt.ylabel(column)
    plt.show()
    print(data[column].describe())
```

Ogive/Cumulative frequency for Total no of Clubs, CGPA and Genral Happiness

In [ ]:
```
attributes = ['clubsNo', 'CGPA','happinessGeneral']
for attribute in attributes:
    totalAttribute = data[attribute].value_counts().sort_index().cumsum()
    plt.figure(figsize=(10, 5))
    plt.plot(totalAttribute.index, totalAttribute.values, marker='o', color=pall
    plt.title(f'Ogive for {attribute}')
    plt.xlabel(attribute)

    y_ticks = range(0, 100, 10)
    plt.yticks(y_ticks)
    plt.ylabel('Frequency')
    plt.show()
```

Happiness distribution for data based upon current year and post or under graduation

In [ ]:
```
yearwise_cgpa = data.groupby(['Year','UG/PG'])[['happinessGeneral']].agg(['mean'
print(yearwise_cgpa)
```

Correlation coefficient between different numeric attributes to analyse the dependencies or relation

In [ ]:
```
numeric_data = data.select_dtypes(include=[np.number])
correlation_matrix = numeric_data.corr()

# Selecting the columns of interest
cgpa_correlation = correlation_matrix[['nightOwl','CGPA', 'supportFamily','posTc

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(cgpa_correlation, annot=True, cmap='coolwarm', linewidths=1)
plt.title('Correlation Heat Map')
plt.show()
```

In [ ]:
```
# group average happinessGeneral by Year
grouped = data.groupby(['Year'])['lifeSatisfaction'].agg(['mean', 'max','min', '
grouped
# grouped.plot(kind='bar', edgecolor='black')
```

In [ ]:
```
data['happinessGeneral'].describe()
```

Central Limit Theorem

The Central Limit Theorem (CLT) states that the distribution of the sample means (or sums) of a large number of independent, identically distributed variables will be approximately normally distributed, regardless of the original distribution of the variables. This approximation improves as the sample size increases.

Verifying CLT on happiness, religion attrinute

In this case, we are applying the CLT to the happiness and religion attributes by taking multiple random samples from the dataset, computing their means, and plotting the distribution of these sample means. As per the CLT, this distribution should approximate a normal distribution, even if the original data is not normally distributed.

In [ ]:
```
# import norms for plotting Normal Distribution
from scipy.stats import norm
```

Central Limit Theorem for Happiness attribute

In [ ]:
```
# Parameters
sample_size = 30
num_samples = 10000

# Take multiple random samples and compute their means
sample_means = []
for _ in range(num_samples):
    sample = data['happinessGeneral'].sample(sample_size, replace=True)
    sample_means.append(sample.mean())

# Plot the distribution of sample means
plt.figure(figsize=(10, 5))
sns.histplot(sample_means, kde=False, stat="density", bins=30, color=pallete1[1]

# Overlay with a normal distribution
mean = np.mean(sample_means)
std_dev = np.std(sample_means)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mean, std_dev)
plt.plot(x, p, 'k', linewidth=2, label='Normal Distribution')

plt.title('Distribution of Sample Means (happinessGeneral)')
plt.xlabel('Sample Mean')
plt.ylabel('Density')
plt.legend()
plt.show()
```

Central limit theorem for Religion attribute

In [ ]:
```
# Parameters
sample_size = 30
num_samples = 10000

# Take multiple random samples and compute their means
sample_means = []
for _ in range(num_samples):
    sample = data['religion'].sample(sample_size, replace=True)
    sample_means.append(sample.mean())

# Plot the distribution of sample means
plt.figure(figsize=(10, 5))
sns.histplot(sample_means, kde=False, stat="density", bins=30, color=pallete1[1]

# Overlay with a normal distribution
mean = np.mean(sample_means)
std_dev = np.std(sample_means)
```

```
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mean, std_dev)
plt.plot(x, p, 'k', linewidth=2, label='Normal Distribution')

plt.title('Distribution of Sample Means (Religion)')
plt.xlabel('Sample Mean')
plt.ylabel('Density')
plt.legend()
plt.show()
```

As the approximation to normal increases with no of samples we tried to verify this by plotting the sample means distribution for sample sizes of 100,1000 and 10000.