Instructions: Lab test 2023 - CZ4025

- The maximum total points you can receive in this lab quiz is capped at 30.
- You have ***2 hours*** to complete the tasks.
- You may use web resources to scout for programming library documents, conceptual background, etc. in solving the tasks. You can also use AI tools such as LLMs. However, ***do not plagiarize*** code. Write your own code instead. I might run your submission through a plagiarism checking software and/or carry out manual checks to detect and penalize plagiarism.
- Though you will have internet connection, ***DO NOT communicate with anyone***, within the class or outside. This lab test is in part trust-based and depends on a honour system. Thus, if you are caught cheating/communicating with others (or if you are reported to do so by a classmate with credible evidence to said effect), you will be given 0 marks for the whole course, in effect, you will fail the course. Furthermore, I will report you for disciplinary actions by the university for cheating.
- Upload a .zip file containing your jupyter notebook (with all the outputs) and a HTML and/or PDF (also with the outputs) on NTUlearn through the applicable assignment link. The files should be named ***YourName .zip, .ipynb and .pdf/.html*** respectively.
- If you use any AI services such as ChatGPT, please provide a clear description (at the end of this notebook) of how you used it, e.g., what prompts were used, how the response (if possible, share screenshots) was used in your solution.
- Please do NOT share the test questions or solutions with anyone else, even after this semester or even your studies at NTU are over. This does not however prevent you from discussing the solutions **after the test is over**, among fellow students from the same cohort.

In [1]:
```
# import main modules
import pandas as pd
import numpy as np
```

Problem 1: **(4 points)**

Consider the provided ExtortionEmailCollection.html file. Create a list of Bitcoin addresses that can be found in this html file.

You may alternatively directly use https://www.u.tsukuba.ac.jp/phishing-collection/ (https://www.u.tsukuba.ac.jp/phishing-collection/) from which the html file was created.

An accompanying BitcoinAddressFormats.pdf file contains some information regarding various Bitcoin address formats. Feel free to use other online resources to find more information about Bitcoin address formats.

In [2]:
```
from tqdm.notebook import tqdm
import pickle
import json
import re
import requests
from bs4 import BeautifulSoup

# Creating PrettyPrinter Instance
import pprint
pp = pprint.PrettyPrinter(indent=2)
```

In [3]:
```
# Creating a regex function to extract bitcoin addreses
def extract_bitcoin_addresses(text):

    pattern = r'[13][a-km-zA-HJ-NP-Z0-9]{26,35}'

    bitcoin_address = re.findall(pattern, text)

    return ' '.join(bitcoin_address)
```

In [4]:
```
extract_bitcoin_addresses('15wz4Cccpwf7UKz3C6VWoAM4fJi6gKqvrR')
```

Out[4]:
```
'15wz4Cccpwf7UKz3C6VWoAM4fJi6gKqvrR'
```

In [5]:
```
# Your code for Problem 1 here
url = "https://www.u.tsukuba.ac.jp/phishing-collection/"
response = requests.get(url)
html_content = response.text

soup = BeautifulSoup(html_content, "html.parser")

# Find all blockquote elements that consist of each container information
blockquotes = soup.find_all("blockquote")

bitcoin_addresses = []

# Checking for bitcoin addresses
for blockquote in blockquotes:

    blockquote_text = blockquote.find('pre').text
    blockquote_text_lower = blockquote_text.lower()

    # Check if the blockquote contains Bitcoin-related information
    if "btc" in blockquote_text_lower or 'bitcoin' in blockquote_text_lower:

        target = extract_bitcoin_addresses(blockquote_text)
        if target:
            bitcoin_addresses.append(target)
            print("Extracted Bitcoin Address:", target)
        print("Bitcoin Address Found")
```

```
Extracted Bitcoin Address: 15wz4Cccpwf7UKz3C6VWoAM4fJi6gKqvrR
Bitcoin Address Found
Extracted Bitcoin Address: 15wz4Cccpwf7UKz3C6VWoAM4fJi6gKqvrR
Bitcoin Address Found
Extracted Bitcoin Address: 18wUUSghRQJ2FJoBY9TuE9xqPooSqCvTXX
Bitcoin Address Found
Extracted Bitcoin Address: 1H1K8MfLEJgjCCfDEkTJmv9GJjD3XzEFGR
Bitcoin Address Found
Extracted Bitcoin Address: 15uBUPv1gzyDRu9psWEujr76XqjiTLZqk4
Bitcoin Address Found
Extracted Bitcoin Address: 1M3uh3QNTxVsK1MqR4cBdqajojUixCiwwq
Bitcoin Address Found
Extracted Bitcoin Address: 1Hbfkn3aPByGQFJRqS9ce26qQNEpG9rp4T
Bitcoin Address Found
Extracted Bitcoin Address: 1PS1N19snfwSE9WTjrveoKgEYEuJM99qR9
Bitcoin Address Found
```

In [6]:
```
# Print the information stored in the bitcoin_list
for address in bitcoin_addresses:
    print(address)
```

```
15wz4Cccpwf7UKz3C6VWoAM4fJi6gKqvrR
15wz4Cccpwf7UKz3C6VWoAM4fJi6gKqvrR
18wUUSghRQJ2FJoBY9TuE9xqPooSqCvTXX
1H1K8MfLEJgjCCfDEkTJmv9GJjD3XzEFGR
15uBUPv1gzyDRu9psWEujr76XqjiTLZqk4
1M3uh3QNTxVsK1MqR4cBdqajojUixCiwwq
1Hbfkn3aPByGQFJRqS9ce26qQNEpG9rp4T
1PS1N19snfwSE9WTjrveoKgEYEuJM99qR9
```

Problem 2: **(5 points)**

Consider the weight-height.csv file, which has three columns indicating individuals' gender, height and weight.

While collecting the data, the respondents were advised to report their weight and height correctly, however they were to report their gender through a randomization process, thus providing privacy through plausible deniability.

Consider the following random response mechanism. The respondent flips a balanced coin. If the coin flip results in a HEAD, then the respondent provides the TRUE answer regarding their gender. If the coin flip results in a TAIL, then the respondent provides Male or Female as responses with equal probability, i.e., 0.5.

Determine an estimate for the actual number of male and female respondents, and explain your approach in the discussion place-holder below the code place-holder.

In [7]:
```
# Your code for Problem 2 here
df = pd.read_csv('weight-height.csv')
df
```

Out[7]:

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801
...	...	...	...
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461
9999	Female	61.944246	113.649103

10000 rows × 3 columns

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Gender  10000 non-null  object 
 1   Height  10000 non-null  float64
 2   Weight  10000 non-null  float64
dtypes: float64(2), object(1)
memory usage: 234.5+ KB
```

In [9]:
```
male_respondents = df[df['Gender'] == 'Male']
female_respondents = df[df['Gender'] == 'Female']
print(len(male_respondents), len(female_respondents))
```

```
5000 5000
```

In [10]:
```
def estimate_genders(total_respondents, reported_males):

    # Solving for M and F
    actual_males = (reported_males - 0.25 * total_respondents) / 0.5
    actual_females = total_respondents - actual_males

    return actual_males, actual_females

num_respondents = 10000
male_respondents = 5000
estimate_genders(num_respondents, male_respondents)
```

Out[10]:
```
(5000.0, 5000.0)
```

Discussion place-holder: Your justification and the analysis of your result is to be discussed here.

Answer:

Probabilities based on random response mechanism

- P(Report Male | Male) = 0.5 * 1 + 0.5 * 0.5 = 0.75
- P(Report Male | Female) = 0.5 * 0 + 0.5 * 0.5 = 0.25

Based on the equations above, we get:

- R_M = 0.75M + 0.25F (1)
- M + F = N (2)

Hence, I created a function to represent equation (1) in the probabilities and using (2) to get the actual females by taking the total respondents subtract the estimated number of male respondents using the probabilites calculated

---

Problem 3: **(8 points)**

Consider the list of countries elected as members of the security counil: https://www.un.org/securitycouncil/content/countries-elected-members (https://www.un.org/securitycouncil/content/countries-elected-members)

Consider also the list of countries that have never been in the security council: https://www.un.org/securitycouncil/content/countries-never-elected-members-security-council (https://www.un.org/securitycouncil/content/countries-never-elected-members-security-council)

Create a (set of) visualization(s) that encode in a consolicated manner the various kinds of information available across these two webpages.

Consider (and explain in the accompanying Discussion place-holder) why you have chosen a given visualization technique, and how you have ensured it has low lie-factor, high data-ink ratio while also meeting accessibility and aesthetics requirements.

In [11]:
```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
```

In [12]:
```
elected_url = 'https://www.un.org/securitycouncil/content/countries-elected-members'
non_elected_url = 'https://www.un.org/securitycouncil/content/countries-never-elected-members-security-council'
```

Below, I have used selenium to scrape the whole chunk of countries so that it will be easier for me to store the data reducing time spent to copy paste from the actual website.

Elected Countries

In [13]:
```
# Your code for Problem 3

# Using Selenium instead of using Requests due to UN Security Page
driver = webdriver.Chrome()
driver.get(elected_url)
elected_countries = []

page_source = driver.page_source
soup = BeautifulSoup(page_source, 'html.parser')

strong_container = soup.find_all('strong')

for element in strong_container:
    elected_countries.append(element.get_text())
    print(element.get_text())

driver.quit()
```

```
India
Indonesia
Iran (Islamic Republic of)
Iraq
Ireland
Italy
Ivory Coast
Jamaica
Japan
Jordan
Kazakhstan
Kenya
Kuwait
Lebanon
Liberia
Libyan Arab Jamahiriya
Lithuania
Luxembourg
Madagascar
Malaysia
```

In [14]:
```
text_1 = """
Albania
Algeria
Angola
Argentina
Australia
Austria
Azerbaijan
Bahrain
Bangladesh
Belarus
Belgium
Benin
Bolivia
Bosnia and Herzegovina
Botswana
Brazil
Bulgaria
Burkina Faso
Burundi
Cabo Verde
Cameroon
Canada
Ceylon
Chad
Chile
China
Colombia
Congo
Costa Rica
Côte d'Ivoire
Croatia
Cuba
Czech Republic
Czechoslovakia
Democratic Republic of the Congo
Denmark
Djibouti
Dominican Republic
Ecuador
Egypt
Equatorial Guinea
Estonia
Ethiopia
Finland
France
Gabon
Gambia
German Democratic Republic
Germany
Ghana
Greece
Guatemala
Guinea
Guinea-Bissau
Guyana
Honduras
Hungary
India
Indonesia
Iran (Islamic Republic of)
Iraq
Ireland
Italy
Ivory Coast
Jamaica
Japan
Jordan
Kazakhstan
Kenya
Kuwait
Lebanon
Liberia
Libyan Arab Jamahiriya
Lithuania
Luxembourg
Madagascar
Malaysia
Mali
Malta
Mauritania
Mauritius
Mexico
Morocco
Mozambique
Namibia
Nepal
the Netherlands
New Zealand
Nicaragua
```

```
Niger
Nigeria
Norway
Oman
Pakistan
Panama
Paraguay
Peru
Philippines
Poland
Portugal
Qatar
Republic of Korea
Romania
Russian Federation
Rwanda
Saint Vincent and the Grenadines
Saudi Arabia
Senegal
Sierra Leone
Singapore
Slovakia
Slovenia
Somalia
South Africa
Spain
Sri Lanka
Sudan
Sweden
Switzerland
Syrian Arab Republic
Thailand
Togo
Trinidad and Tobago
Tunisia
Türkiye
Uganda
Ukraine
Union of Soviet Socialist Republics
United Arab Emirates
United Arab Republic
United Kingdom of Great Britain and Northern Ireland
United Republic of Tanzania
United States of America
Uruguay
Venezuela (Bolivarian Republic of)
Viet Nam
Yemen
Yugoslavia
Zaire
Zambia
Zimbabwe
"""

elected_countries = text_1.strip().split('\n')
print(elected_countries)
```

```
['Albania', 'Algeria ', 'Angola ', 'Argentina ', 'Australia ', 'Austria ', 'Azerbaijan ', 'Bahrain ', 'Banglad
esh ', 'Belarus ', 'Belgium ', 'Benin ', 'Bolivia ', 'Bosnia and Herzegovina ', 'Botswana ', 'Brazil ', 'Bulga
ria ', 'Burkina Faso ', 'Burundi ', 'Cabo Verde ', 'Cameroon ', 'Canada ', 'Ceylon', 'Chad ', 'Chile ', 'Chin
a', 'Colombia ', 'Congo ', 'Costa Rica ', "Côte d'Ivoire ", 'Croatia ', 'Cuba ', 'Czech Republic ', 'Czechoslo
vakia ', 'Democratic Republic of the Congo ', 'Denmark ', 'Djibouti ', 'Dominican Republic ', 'Ecuador ', 'Egy
pt ', 'Equatorial Guinea ', 'Estonia', 'Ethiopia ', 'Finland ', 'France', 'Gabon ', 'Gambia ', 'German Democra
tic Republic', 'Germany ', 'Ghana ', 'Greece ', 'Guatemala ', 'Guinea ', 'Guinea-Bissau ', 'Guyana ', 'Hondura
s ', 'Hungary ', 'India ', 'Indonesia ', 'Iran (Islamic Republic of) ', 'Iraq ', 'Ireland ', 'Italy ', 'Ivory
Coast ', 'Jamaica ', 'Japan ', 'Jordan ', 'Kazakhstan ', 'Kenya ', 'Kuwait ', 'Lebanon ', 'Liberia ', 'Libyan
Arab Jamahiriya ', 'Lithuania ', 'Luxembourg ', 'Madagascar ', 'Malaysia ', 'Mali ', 'Malta ', 'Mauritania ',
'Mauritius ', 'Mexico ', 'Morocco ', 'Mozambique', 'Namibia ', 'Nepal ', 'the Netherlands ', 'New Zealand ',
'Nicaragua ', 'Niger ', 'Nigeria ', 'Norway ', 'Oman ', 'Pakistan ', 'Panama ', 'Paraguay ', 'Peru ', 'Philipp
ines ', 'Poland ', 'Portugal ', 'Qatar ', 'Republic of Korea ', 'Romania ', 'Russian Federation', 'Rwanda ',
'Saint Vincent and the Grenadines', 'Saudi Arabia', 'Senegal ', 'Sierra Leone ', 'Singapore ', 'Slovakia ', 'S
lovenia ', 'Somalia ', 'South Africa ', 'Spain ', 'Sri Lanka ', 'Sudan ', 'Sweden ', 'Switzerland', 'Syrian Ar
ab Republic ', 'Thailand ', 'Togo ', 'Trinidad and Tobago ', 'Tunisia ', 'Türkiye', 'Uganda ', 'Ukraine ', 'Un
ion of Soviet Socialist Republics ', 'United Arab Emirates ', 'United Arab Republic ', 'United Kingdom of Grea
t Britain and Northern Ireland', 'United Republic of Tanzania ', 'United States of America', 'Uruguay ', 'Vene
zuela (Bolivarian Republic of) ', 'Viet Nam ', 'Yemen ', 'Yugoslavia ', 'Zaire', 'Zambia ', 'Zimbabwe']
```

Non-Elected Countries

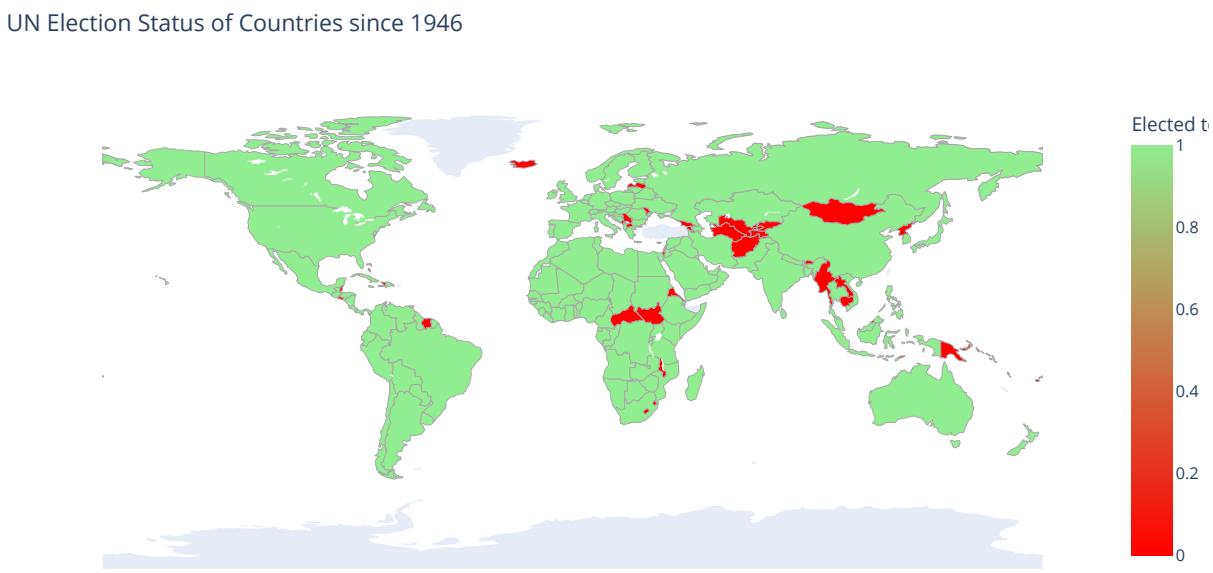
In [15]:
```
# Using Selenium instead of using Requests due to UN Security Page
driver = webdriver.Chrome()
driver.get(non_elected_url)

page_source = driver.page_source
soup = BeautifulSoup(page_source, 'html.parser')

li_elements = soup.find_all('li')

for li_element in li_elements:
    print(li_element.get_text())
```

```
Comoros
Cyprus
Democratic People's Republic of Korea
Dominica
El Salvador
Eritrea
Fiji
Georgia
Grenada
Haiti
Iceland
Israel
Kiribati
Kyrgyzstan
Lao People's Democratic Republic
Latvia
Lesotho
Liechtenstein
Malawi
Maldives
```

In [16]:
```
text_2 = """
Afghanistan
Andorra
Antigua and Barbuda
Armenia
Bahamas
Barbados
Belize
Bhutan
Brunei Darussalam
Cambodia
Central African Republic
Comoros
Cyprus
Democratic People's Republic of Korea
Dominica
El Salvador
Eritrea
Fiji
Georgia
Grenada
Haiti
Iceland
Israel
Kiribati
Kyrgyzstan
Lao People's Democratic Republic
Latvia
Lesotho
Liechtenstein
Malawi
Maldives
Marshall Islands
Micronesia (Federated States of)
Monaco
Mongolia
Montenegro
Myanmar
Nauru
North Macedonia
Palau
Papua New Guinea
Republic of Moldova
Saint Kitts and Nevis
Saint Lucia
Samoa
San Marino
Sao Tome and Principe
Serbia
Seychelles
Solomon Islands
South Sudan
Suriname
Swaziland
Tajikistan
Timor-Leste
Tonga
Turkmenistan
Tuvalu
Uzbekistan
Vanuatu
"""

non_elected_countries = text_2.strip().split('\n')
print(non_elected_countries)
```

```
['Afghanistan', 'Andorra', 'Antigua and Barbuda', 'Armenia', 'Bahamas', 'Barbados', 'Belize', 'Bhutan', 'Brune
i Darussalam', 'Cambodia', 'Central African Republic', 'Comoros', 'Cyprus', "Democratic People's Republic of K
orea", 'Dominica', 'El Salvador', 'Eritrea', 'Fiji', 'Georgia', 'Grenada', 'Haiti', 'Iceland', 'Israel', 'Kiri
bati', 'Kyrgyzstan', "Lao People's Democratic Republic", 'Latvia', 'Lesotho', 'Liechtenstein', 'Malawi', 'Mald
ives', 'Marshall Islands', 'Micronesia (Federated States of)', 'Monaco', 'Mongolia', 'Montenegro', 'Myanmar',
'Nauru', 'North Macedonia', 'Palau', 'Papua New Guinea', 'Republic of Moldova', 'Saint Kitts and Nevis', 'Sain
t Lucia', 'Samoa', 'San Marino', 'Sao Tome and Principe', 'Serbia', 'Seychelles', 'Solomon Islands', 'South Su
dan', 'Suriname', 'Swaziland', 'Tajikistan', 'Timor-Leste', 'Tonga', 'Turkmenistan', 'Tuvalu', 'Uzbekistan',
'Vanuatu']
```

In [17]:
```
import plotly.express as px
import plotly.graph_objs as go

# Create a DataFrame with all countries and their election status
countries_df = pd.DataFrame({
    'Country': elected_countries + non_elected_countries,
    'Elected': ['Yes'] * len(elected_countries) + ['No'] * len(non_elected_countries)
})

fig = go.Figure(data=go.Choropleth(
    locations=countries_df['Country'],
    locationmode='country names',
    z=countries_df['Elected'].astype('category').cat.codes,
    text=countries_df['Country'],
    colorscale=['red', 'lightgreen'],
    colorbar_title='Elected to UN',
    marker_line_color='darkgray',
    marker_line_width=0.5,
))

fig.update_layout(
    title_text='UN Election Status of Countries since 1946',
    geo=dict(
        showframe=False,
        showcoastlines=False,
        projection_type='equirectangular'
    ),
)

fig.show()
```



Discussion place-holder for Problem 3.

Question: Consider (and explain in the accompanying Discussion place-holder) why you have chosen a given visualization technique, and how you have ensured it has low lie-factor, high data-ink ratio while also meeting accessibility and aesthetics requirements.

Answer:

- I have chosen the given visualisation techniques as I think that showing the data in a choropleth map allows a whole overview around the world to show countries that have been elected and those who have not been elected since 1946.
- I have ensured a high data-ink ratio by encoding the countries who have not been elected to be a dark shade of red to place emphasis on these countries and a lighter tone of green on countries who have been elected so that the humans perceptual will be on the countries who have not been elected as they are the minority.
- Choropleth maps typically do not have a "low-lie factor." However, i feel that in this case this can be omited as the functionality of plotly allows zoom-in features to certain part of the maps. The use of colour gradients and patterns based on the Equirectangular Projection of the world map also helps with identifying the election sttays of specific countries of the world
- Lastly, the plot is aesthetic as the Plotly libraries allows us to customise the colour palette such that it is pleasing to the eyes and additionally, the use of user interactivity on the plot makes it easier to visualise.

Problem 4: **(3+5=8 points: This is a multi-part question)**

Consider the data provided in Salary_Data_gender_and_race.csv

Problem 4.1: For the employees in the USA, use a suitable visualization tool to compare the salary of people whose race is identified as "White" versus those who are not "White".

Problem 4.2: Using sutable test, determine whether there is any statistically significant (with at least 95% confidence) difference in salary across Male and Female employees identified as non-White in the USA, for the provided dataset.

In the explanation part, clearly state your Null and alternate hypothesis, explain which test you chose to use, why it is suitable, and elaborate the interpretation of the test result.

In [18]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [19]:
```
# Your code for Problem 4.1
salary_df = pd.read_csv('Salary_Data_gender_and_race.csv')
salary_df.head()
```

Out[19]:

	Unnamed: 0	Age	Gender	Education Level	Job Title	Years of Experience	Salary	Country	Race
0	0	32	Male	Bachelor's	Software Engineer	5.0	$90,000.00	UK	White
1	1	28	Female	Master's	Data Analyst	3.0	$65,000.00	USA	Hispanic
2	2	45	Male	PhD	Senior Manager	15.0	$150,000.00	Canada	White
3	3	36	Female	Bachelor's	Sales Associate	7.0	$60,000.00	USA	Hispanic
4	4	52	Male	Master's	Director	20.0	$200,000.00	USA	Asian

In [20]:
```
salary_df['Race'].unique()
```

Out[20]:
```
array(['White', 'Hispanic', 'Asian', 'Korean', 'Chinese', 'Australian',
       'Welsh', 'African American', 'Mixed', 'Black'], dtype=object)
```

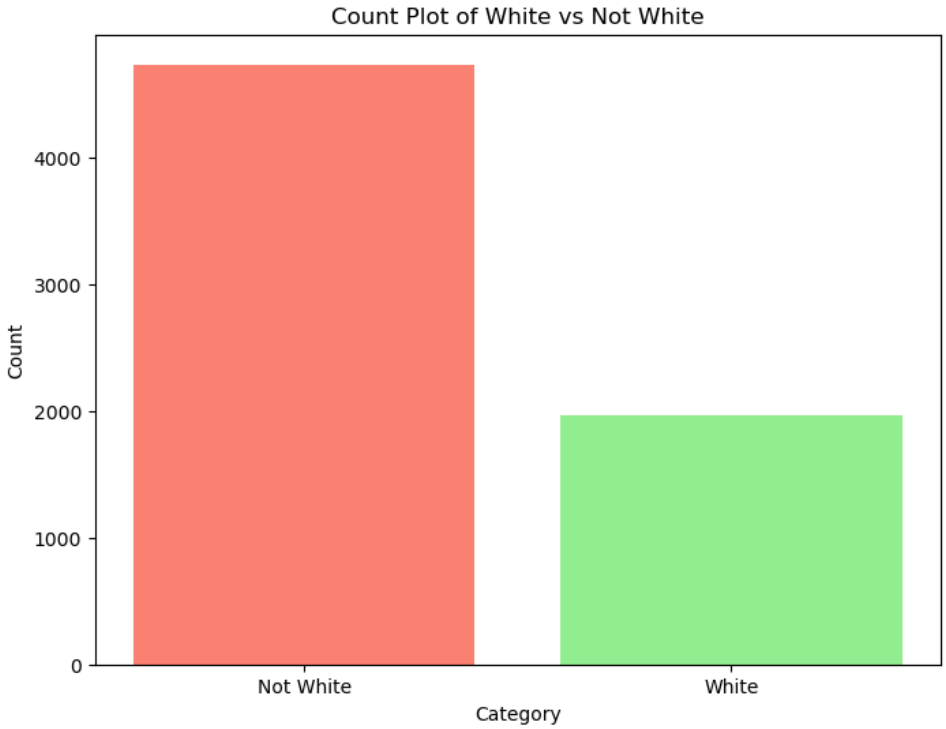
In [21]:
```
# Replace categories that are not 'White' with 'Not White'
salary_df['Race Category'] = salary_df['Race'].apply(lambda x: 'Not White' if x != 'White' else x)
salary_df
```

Out[21]:

	Unnamed: 0	Age	Gender	Education Level	Job Title	Years of Experience	Salary	Country	Race	Race Category
0	0	32	Male	Bachelor's	Software Engineer	5.0	$90,000.00	UK	White	White
1	1	28	Female	Master's	Data Analyst	3.0	$65,000.00	USA	Hispanic	Not White
2	2	45	Male	PhD	Senior Manager	15.0	$150,000.00	Canada	White	White
3	3	36	Female	Bachelor's	Sales Associate	7.0	$60,000.00	USA	Hispanic	Not White
4	4	52	Male	Master's	Director	20.0	$200,000.00	USA	Asian	Not White
...	...	...	...	...	...	...	...	...	...	...
6695	6699	49	Female	PhD	Director of Marketing	20.0	$200,000.00	UK	Mixed	Not White
6696	6700	32	Male	High School	Sales Associate	3.0	$50,000.00	Australia	Australian	Not White
6697	6701	30	Female	Bachelor's Degree	Financial Manager	4.0	$55,000.00	China	Chinese	Not White
6698	6702	46	Male	Master's Degree	Marketing Manager	14.0	$140,000.00	China	Korean	Not White
6699	6703	26	Female	High School	Sales Executive	1.0	$35,000.00	Canada	Black	Not White

6700 rows × 10 columns

In [22]:
```
category_counts = salary_df['Race Category'].value_counts()

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

colors = ['lightgreen' if category == 'White' else 'salmon' for category in category_counts.index]

plt.bar(category_counts.index, category_counts.values, color=colors)

plt.xlabel('Category')
plt.ylabel('Count')
plt.title('Count Plot of White vs Not White')

plt.show()
```



In [23]:
```
# Your code for Problem 4.2
from scipy import stats

# First, check whether distributions of Non White Females and Males are normally distributed
filtered_data = salary_df.copy()
filtered_data = filtered_data[(filtered_data['Country'] == 'USA') & (filtered_data['Race Category'] == 'Not White

# Remove non-numeric characters like $ and commas
filtered_data['Salary'] = filtered_data['Salary'].replace('[\$,]', '', regex=True).astype(float)

male_salaries = filtered_data[filtered_data['Gender'] == 'Male']['Salary']
female_salaries = filtered_data[filtered_data['Gender'] == 'Female']['Salary']

# Shapiro-Wilk Test
w_statistic_male, p_value_male = stats.shapiro(male_salaries)
w_statistic_female, p_value_female = stats.shapiro(female_salaries)

print(f'Male Salaries - W-statistic: {w_statistic_male}, P-value: {p_value_male}')
print(f'Female Salaries - W-statistic: {w_statistic_female}, P-value: {p_value_female}')
```

```
Male Salaries - W-statistic: 0.9530461430549622, P-value: 3.0192868604589362e-12
Female Salaries - W-statistic: 0.964576244354248, P-value: 4.774281414654524e-09
```

Both P-values <<< 0.05, hence both distributions are not normal, hence Two-tailed T-test cannot be done.

Hence, a Mann-Whitney U test will be conducted on the two distributions and the reasons being

- I am comparing two independent samples of non-whites male and female salary data
- This test does not assume normal distribution of the data, making it suitable in this case as the data of both distributions are not normal (proven above)
- The Mann-Whitney U test compares the medians of two groups, providing a good alternative when the assumptions for the t-test are not met.

In [24]:
```
from scipy.stats import mannwhitneyu

# Perform the Mann-Whitney U test
u_statistic, p_value = mannwhitneyu(male_salaries, female_salaries)

print(f'U-statistic: {u_statistic}, P-value: {p_value}')
```

```
U-statistic: 145635.5, P-value: 2.2931749325015127e-05
```

Discussion place-holder for Problem 4.2

- Null hypothesis H_0: There is no difference in salary distribution between male and female non-White employees.
- Alternative hypothesis H_1: There is a Significant difference in salary distribution between male and female non-White employees.
- Based on the Mann-Whitney U Test, the p-value = 2.2931e-05 <<< 0.05, it suggests that there is a statistically significant difference in salary distributions between male and female non-White employees.

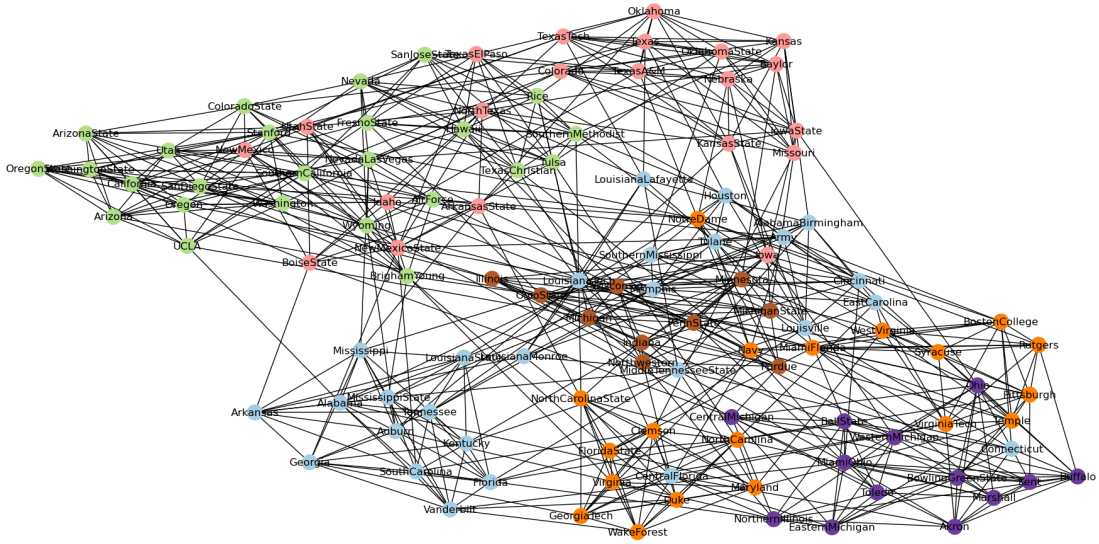
Problem 5: **(5 points)**

Consider the below interaction graph based on games played by various footbal teams. Use a suitable community detection (clustering) algorithm to determine the number of meaningful communities, and membership of the football teams in those communities. Provide the final answer in the form of a list of lists, where each sublist indicates the membership of the individual community (by including the name of the teams). Provide a very brief explanation for the rational of your choice of the community detection algorithm.

In [25]:
```
import urllib.request
import io
import zipfile

import matplotlib.pyplot as plt
import networkx as nx

url = "http://www-personal.umich.edu/~mejn/netdata/football.zip"

sock = urllib.request.urlopen(url)  # open URL
s = io.BytesIO(sock.read())  # read into BytesIO "file"
sock.close()

zf = zipfile.ZipFile(s)  # zipfile object
txt = zf.read("football.txt").decode()  # read info file
gml = zf.read("football.gml").decode()  # read gml data
gml = gml.split("\n")[1:]
G = nx.parse_gml(gml)  # parse gml data

# print name of all the teams
for n in G.nodes():
    print(f"{n:20}")
```

```
BrighamYoung
FloridaState
Iowa
KansasState
NewMexico
TexasTech
PennState
SouthernCalifornia
ArizonaState
SanDiegoState
Baylor
NorthTexas
NorthernIllinois
Northwestern
WesternMichigan
Wisconsin
Wyoming
Auburn
Akron
```

In [26]:
```
communities = list(nx.algorithms.community.greedy_modularity_communities(G))
#!pip install python-louvain
colors = []
for node in G:
    for i in range(len(communities)):
        if node in communities[i]:
            colors.append(i)

fig = plt.figure(figsize=(20,10))
pos = nx.spring_layout(G)
nx.draw(G,pos, with_labels = 'true',cmap=plt.cm.Paired, node_color=colors)
plt.show()
```



In [27]:
```
# Convert each frozenset to a list and store in a new list
communities_list = [list(community) for community in communities]

# Print each community
for i, community in enumerate(communities_list):
    print(f"Community {i + 1}:")
    print(", ".join(community))
    print("==========")
```

```
Community 1:
LouisianaMonroe, MississippiState, Tulane, Army, Auburn, SouthernMississippi, Tennessee, Florida, LouisianaLaf
ayette, MiddleTennesseeState, Alabama, LouisianaState, Mississippi, CentralFlorida, Louisville, Houston, Alaba
maBirmingham, Memphis, Arkansas, SouthCarolina, Georgia, Connecticut, Kentucky, Cincinnati, LouisianaTech, Eas
tCarolina, Vanderbilt
==========
Community 2:
Stanford, Arizona, NevadaLasVegas, Oregon, ColoradoState, Rice, Nevada, UCLA, California, ArizonaState, Southe
rnMethodist, WashingtonState, OregonState, Utah, SanJoseState, Tulsa, Wyoming, Washington, SanDiegoState, Hawa
ii, BrighamYoung, FresnoState, TexasChristian, SouthernCalifornia, AirForce
==========
Community 3:
NewMexicoState, IowaState, Iowa, Oklahoma, NewMexico, TexasA&M, Kansas, BoiseState, NorthTexas, ArkansasState,
Missouri, Nebraska, Texas, KansasState, OklahomaState, Idaho, Colorado, TexasTech, TexasElPaso, Baylor, UtahSt
ate
==========
Community 4:
Pittsburgh, NorthCarolina, Navy, VirginiaTech, BostonCollege, FloridaState, NorthCarolinaState, WestVirginia,
Syracuse, Rutgers, WakeForest, Maryland, NotreDame, Temple, Clemson, Virginia, GeorgiaTech, MiamiFlorida, Duke
==========
Community 5:
BowlingGreenState, Akron, Ohio, MiamiOhio, NorthernIllinois, BallState, WesternMichigan, Kent, Buffalo, Marsha
ll, CentralMichigan, EasternMichigan, Toledo
==========
Community 6:
OhioState, Michigan, Indiana, Illinois, MichiganState, Purdue, PennState, Wisconsin, Northwestern, Minnesota
==========
```

Discussion place-holder for Problem 5 Answer:

- I have chosen to use Louvain algorithm to form the clusters for each community of football teams as there are large modularity involved in the graph G shown above. It is capable of revealing hierarchical community structures such that it assigns each node to its own community. Thereafter for each node, it considers whether moving it to a neighboring community would increase the modularity.
- The use of Louvain Algorithm also helps in the large scale of network of football teams as it is able to reduce computation time to determine the various clusters

Provide details of use of AI tools such as ChatGPT.

The following information are the resources i have acquired from ChatGPT.

I have refered to ChatGPT for Problems 2 and 3 in the formation of my own code.

Problem 2 Reference to calculation of probability.

python Copy code

```python
import pandas as pd
import numpy as np

# Load the data from the CSV file
data = pd.read_csv('your_data.csv')

# Initialize counts for Male and Female respondents
male_count = 0
female_count = 0

# Iterate through each row in the DataFrame
for index, row in data.iterrows():
    # Simulate the coin flip with a 50% probability for HEAD and TAIL
    coin_flip = np.random.choice(['HEAD', 'TAIL'], p=[0.5, 0.5])

    # If the coin flip is HEAD, take the reported gender as true
    if coin_flip == 'HEAD':
        true_gender = row['Gender']
    # If the coin flip is TAIL, randomly assign Male or Female
    else:
        true_gender = np.random.choice(['Male', 'Female'])

    # Update counts based on the true gender
    if true_gender == 'Male':
        male_count += 1
    elif true_gender == 'Female':
        female_count += 1

# Print the estimated counts
print(f"Estimated number of Male respondents: {male_count}")
print(f"Estimated number of Female respondents: {female_count}")
```