import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sns

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

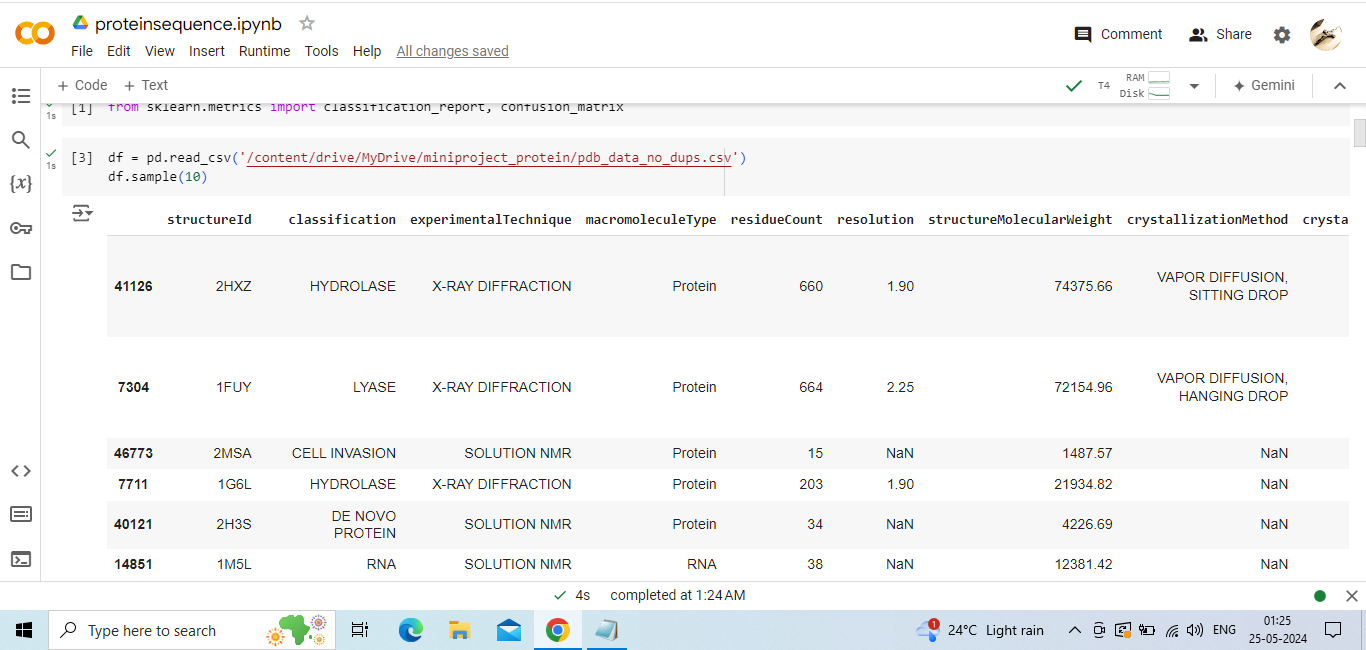
from sklearn.preprocessing import LabelEncoder

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

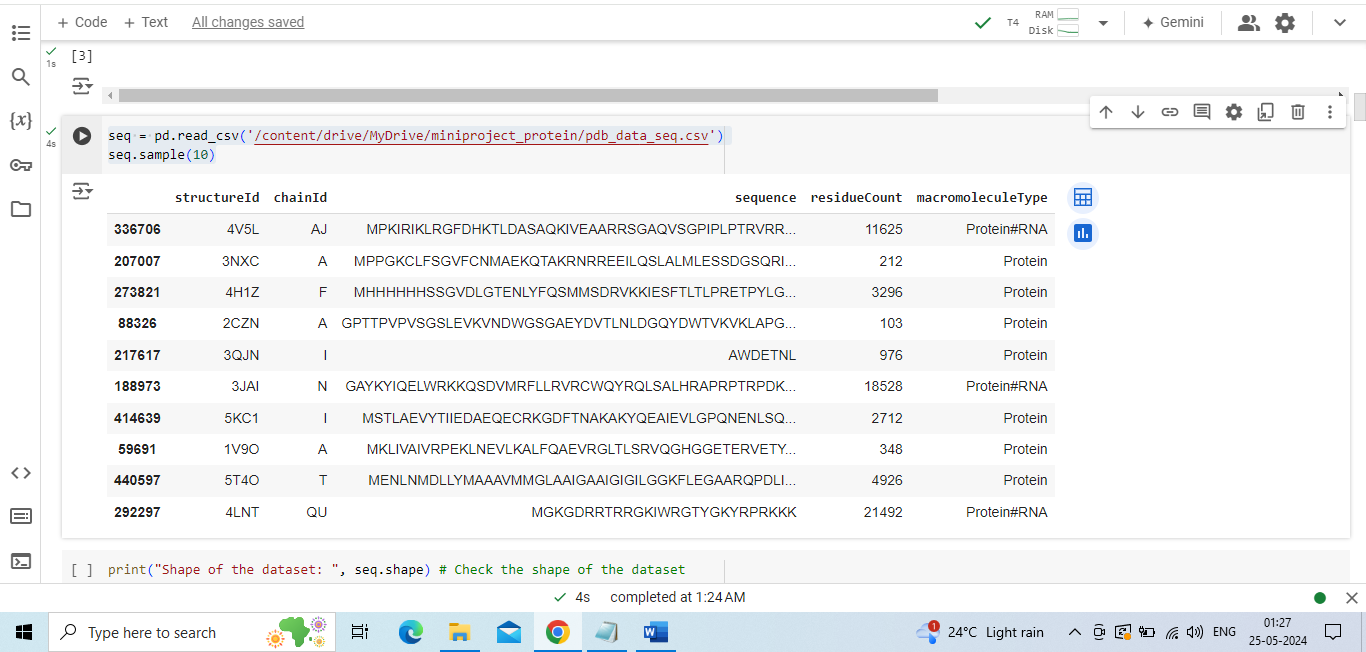
df = pd.read\_csv('/content/drive/MyDrive/miniproject\_protein/pdb\_data\_no\_dups.csv')

df.sample(10)



seq = pd.read\_csv('/content/drive/MyDrive/miniproject\_protein/pdb\_data\_seq.csv')

seq.sample(10)



print("Shape of the dataset: ", seq.shape) # Check the shape of the dataset

print("Data types of the columns: ", seq.dtypes) # Check the data types of the columns

print("Missing values in each column: ", seq.isnull().sum())# Check for missing values

Shape of the dataset: (467304, 5)

Data types of the columns: structureId object

chainId object

sequence object

residueCount int64

macromoleculeType object

dtype: object

Missing values in each column: structureId 0

chainId 10

sequence 28

residueCount 0

macromoleculeType 34817

dtype: int64

seq = seq.dropna()

# Check the distribution of the target variable

print("Distribution of the target variable: ", seq['macromoleculeType'].value\_counts())

# Visualize the distribution of the target variable

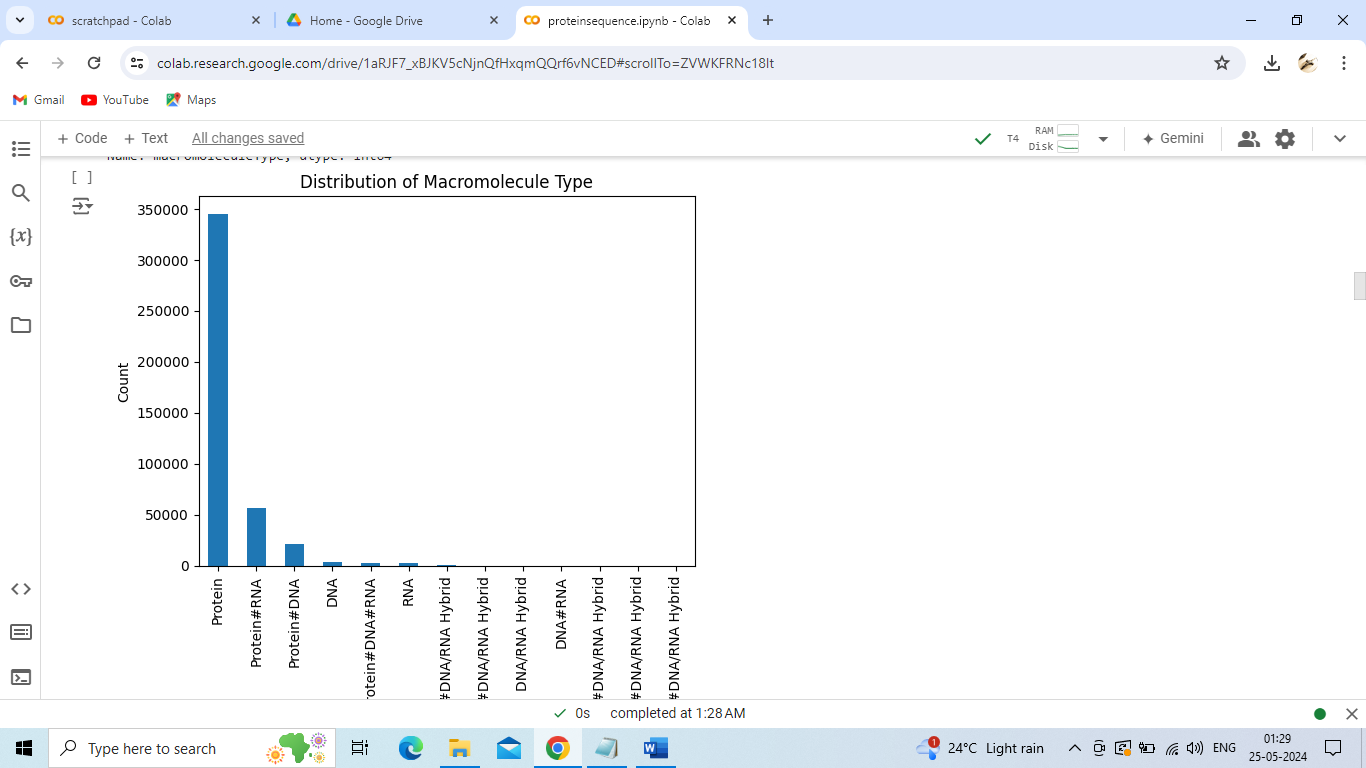
seq['macromoleculeType'].value\_counts().plot(kind='bar')

plt.xlabel('Macromolecule Type')

plt.ylabel('Count')

plt.title('Distribution of Macromolecule Type')

plt.show()

# Convert categorical data to numerical data if necessary

encoder = LabelEncoder()

seq['sequence'] = encoder.fit\_transform(seq['sequence'])

# Select only relevant features

X = seq[['sequence', 'residueCount']]

y = seq['macromoleculeType']

# Split the dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

# Train the KNN model

knn = KNeighborsClassifier(n\_neighbors=5)

knn.fit(X\_train, y\_train)

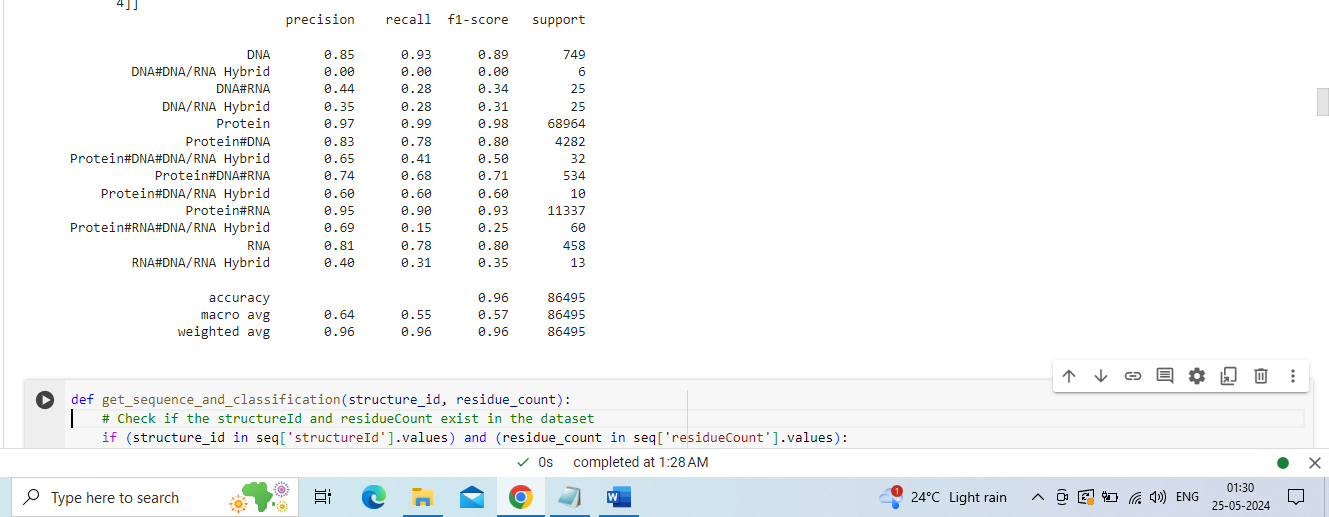
# Make predictions on the test set

y\_pred = knn.predict(X\_test)

# Evaluate the model

print(confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))



def get\_sequence\_and\_classification(structure\_id, residue\_count):

    # Check if the structureId and residueCount exist in the dataset

    if (structure\_id in seq['structureId'].values) and (residue\_count in seq['residueCount'].values):

        # Get the sequence for the given structureId

        sequence = seq.loc[seq['structureId'] == structure\_id, 'sequence'].values[0]

        # Use the trained KNN model to predict the classification

        input\_data = np.array([[sequence, residue\_count]])

        input\_data[:, 0] = encoder.transform(input\_data[:, 0])  # Convert sequence to numerical data

        predicted\_classification = knn.predict(input\_data)

        return sequence, predicted\_classification[0]

    else:

        return "StructureId or residueCount not found in the dataset", None

def get\_sequence\_and\_classification(structure\_id, residue\_count):

    # Convert structure\_id to string, assuming it's a string in your dataset

    structure\_id = str(structure\_id)

    # Check if the structureId and residueCount exist in the dataset

    if (structure\_id in seq['structureId'].values) and (residue\_count in seq['residueCount'].values):

        # Get the sequence for the given structureId

        sequence = seq.loc[(seq['structureId'] == structure\_id) & (seq['residueCount'] == residue\_count), 'sequence'].values

        sequence = sequence[0] if len(sequence) > 0 else None

        # Use the trained KNN model to predict the classification

        input\_data = np.array([[sequence, residue\_count]])

        predicted\_classification = knn.predict(input\_data)

        return sequence, predicted\_classification[0]

    else:

        return "StructureId or residueCount not found in the dataset", None

# Create a hypothetical health dataset

health\_data = {

    'sequence': ['CCGGCGCCGG', 'CGCGAATTCGCG', 'TCC', 'GTGGAATGGAAC','CGCAAATTTGCG'],

    'health\_issue': ['Chance for Heart Disease', 'Muscle Weakness', 'No Issue','No Issue', 'Digestive Problems'],

    'solution': [

        'Increase intake of Omega-3 fatty acids and exercise regularly.',

        'Include more protein and vitamin D in your diet.',

        'No specific health issue detected.',

        'No specific health issue detected.',

        'Consume more fiber and stay hydrated.'

    ]

}

health\_df = pd.DataFrame(health\_data)

def get\_sequence\_and\_classification(structure\_id, residue\_count):

    # Convert structure\_id to string, assuming it's a string in your dataset

    structure\_id = str(structure\_id)

    # Check if the structureId and residueCount exist in the dataset

    if (structure\_id in seq['structureId'].values) and (residue\_count in seq['residueCount'].values):

        # Get the sequence for the given structureId

        sequence\_numerical = seq.loc[(seq['structureId'] == structure\_id) & (seq['residueCount'] == residue\_count), 'sequence'].values

        sequence\_numerical = sequence\_numerical[0] if len(sequence\_numerical) > 0 else None

        # Use the trained KNN model to predict the classification

        input\_data = np.array([[sequence\_numerical, residue\_count]])

        predicted\_classification = knn.predict(input\_data)

        # Inverse transform the numerical sequence back to its original form

        sequence\_original = encoder.inverse\_transform([sequence\_numerical])[0] if sequence\_numerical is not None else None

        # Check for health issues and provide solutions based on the classification

        health\_info = health\_df.loc[health\_df['sequence'] == sequence\_original, ['health\_issue', 'solution']]

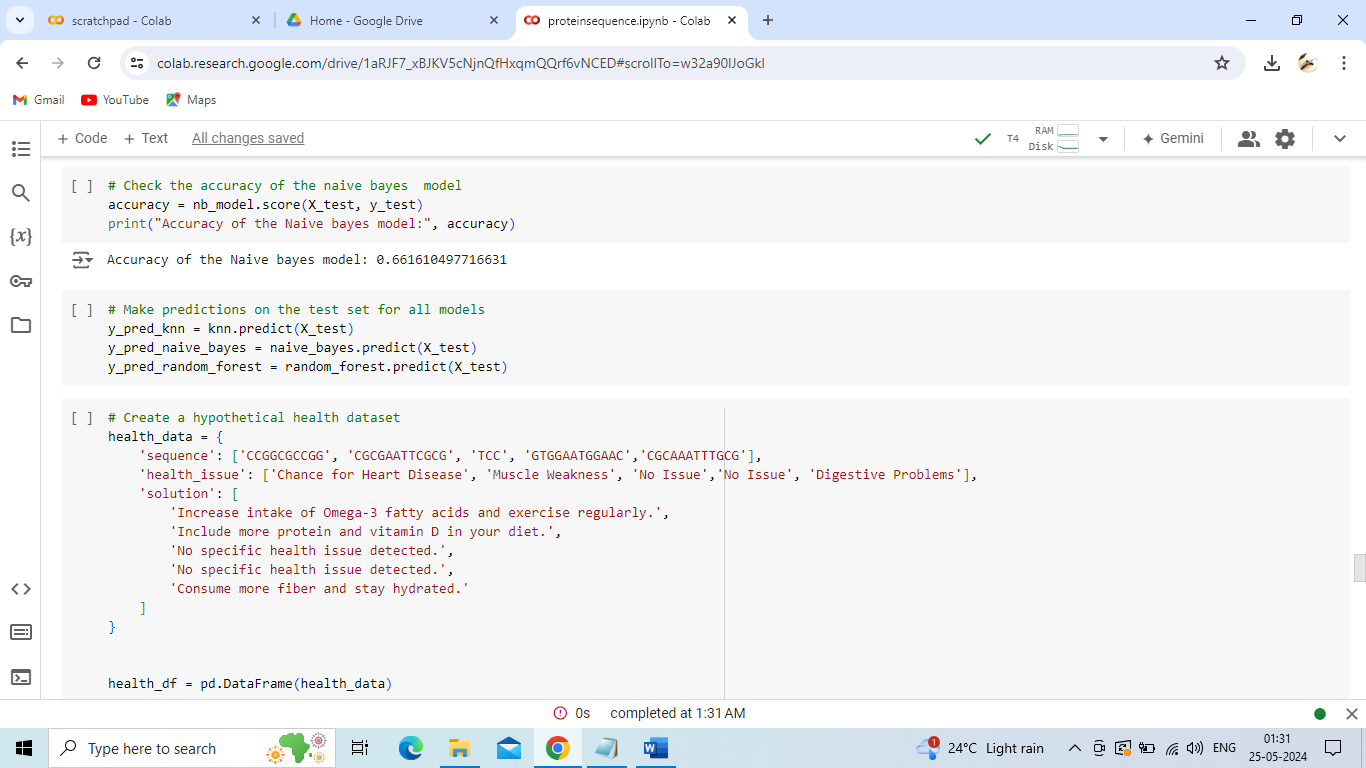
        health\_issue = health\_info['health\_issue'].values[0] if not health\_info.empty else "No health issue found"

        solution = health\_info['solution'].values[0] if not health\_info.empty else ""

        return sequence\_original, predicted\_classification[0], health\_issue, solution

    else:

        return "StructureId or residueCount not found in the dataset", None, None, None



#Example

structure\_id\_to\_search = '100D'

residue\_count\_to\_search = 20

sequence, classification, health\_issue, solution = get\_sequence\_and\_classification(structure\_id\_to\_search, residue\_count\_to\_search)

print("Sequence:", sequence)

print("Classification:", classification)

print("Health Issue:", health\_issue)

print("Solution:", solution)

Sequence: CCGGCGCCGG

Classification: DNA

Health Issue: Chance for Heart Disease

Solution: Increase intake of Omega-3 fatty acids and exercise regularly.



import joblib

# Save the trained KNN model to a file

model\_filename = '/content/drive/MyDrive/miniproject\_protein/protein.h5'

joblib.dump(knn, model\_filename)

# Load the saved model

loaded\_knn\_model = joblib.load(model\_filename)

# Now you can use the loaded\_knn\_model to make predictions

loaded\_y\_pred = loaded\_knn\_model.predict(X\_test)

# Evaluate the loaded model

print("Confusion Matrix for loaded model:")

print(confusion\_matrix(y\_test, loaded\_y\_pred))

print("Classification Report for loaded model:")

print(classification\_report(y\_test, loaded\_y\_pred))

import matplotlib.pyplot as plt

# Model names

models = ['Random Forest', 'KNN', 'Naive Bayes']

# Accuracy scores

accuracies = [98.66, 96.07, 66.16]

# Create a bar chart

plt.figure(figsize=(8, 5))

plt.bar(models, accuracies, color=['green', 'blue', 'red'])

plt.title('Model Accuracy Comparison')

plt.xlabel('Models')

plt.ylabel('Accuracy (%)')

# Display the accuracy values on top of the bars

for i, accuracy in enumerate(accuracies):

    plt.text(i, accuracy + 0.5, f'{accuracy:.2f}%', ha='center', va='bottom')

# Show the plot

plt.show()

# Use t-SNE to reduce dimensionality for visualization

tsne = TSNE(n\_components=2, random\_state=1)

X\_tsne = tsne.fit\_transform(X\_test)

# Plot the results

plt.figure(figsize=(10, 8))

for label in np.unique(y\_test):

    indices = np.where(y\_test == label)

    plt.scatter(X\_tsne[indices, 0], X\_tsne[indices, 1], label=label, alpha=0.5)

plt.title('t-SNE Visualization of Protein Sequence Classification')

plt.xlabel('t-SNE Component 1')

plt.ylabel('t-SNE Component 2')

plt.legend()

plt.show()