Exercise 9: Implement k-NN algorithm to solve classification problem.

```
# importing required python libraries and reading dataset into a dataframe

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from google.colab import drive
drive.mount('/content/gdrive')

# reading dataset from the .csv file and loading into Dataframe
dataset = pd.read_csv('gdrive/My Drive/ml/SUV_Data.csv' )
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
# Print the number of rows (records) and
# columns(attributes) in the X dataset.

dataset.shape
```

```
(400, 5)
```

```
# Otherwise, sample(no. of rows) function can also be used.
# sample() is an inbuilt function of random module in Python.
# It returns a list of items randomly chosen from the sequence.
# It uses random sampling without replacement.

dataset.sample(5)
```

	User ID	Gender	Age	EstimatedSalary	Purchased
364	15654456	Male	42	104000	1
252	15795298	Female	48	134000	1
16	15733883	Male	47	25000	1
143	15783029	Male	30	89000	0
122	15724423	Female	40	75000	0

```
# Extracting Independent variables -- Gender, Age, EstimatedSalary
# User ID is ignored as it does not affect the purchasal

X = pd.DataFrame(dataset.iloc[:, [1, 2, 3]])

# Extracting and target (or dependent) variable -- Purchased

Y = pd.DataFrame(dataset.iloc[:, [4]])
```

```
X
```

	Gender	Age	EstimatedSalary
0	Male	19	19000
1	Male	35	20000
2	Female	26	43000
3	Female	27	57000
4	Male	19	76000
...	...	...	...
395	Female	46	41000
396	Male	51	23000
397	Female	50	20000
398	Male	36	33000
399	Female	49	36000

400 rows × 3 columns

Next steps: Generate code with X View recommended plots

Y

	Purchased
0	0
1	0
2	0
3	0
4	0
...	...
395	1
396	1
397	1
398	0
399	1

400 rows × 1 columns

```
# Check, the unique values of target variable
# Here, target variable is "Purchased"

Y.Purchased.unique()
```

```
array([0, 1])
```

The result shows that there are two unique values for "Purchased" attribute represented as "Purchased = 0", if customer has not purchased SUV car "Purchased = 1", if customer has purchased SUV car

Hence, the given dataset leads to single class classification problem.

```
# Print the information of dataset
# Displays informaton of each attribute - column name,
#                                   non-null count, datatype

X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           400 non-null    object
 1   Age              400 non-null    int64
 2   EstimatedSalary  400 non-null    int64
dtypes: int64(2), object(1)
memory usage: 9.5+ KB
```

it can be observed that there are missing values in the Age attribute.

Check if there are any missing values. If so, they must be handled as part of preprocessing.

isna() is used to detect missing values. It returns the object of bool values for each element in DataFrame that indicates whether an element is an NA value.

mean() is used to calculate mean/average of a given list of numbers.

to_frame() function is used to convert series object to a dataframe.

```
napercentage = X.isna().mean()*100
napercentage.to_frame("% of missing values")
```

	% of missing values
Gender	0.0
Age	0.0
EstimatedSalary	0.0

fillna() method is used to fill missing values in the dataset.

When inplace = True , the data is modified in place, which means it will return nothing and the dataframe is now updated.

When inplace = False , which is the default, then the operation is performed and it returns a copy of the object.

Mode is the value that appears the most in a set of values of an attribute.

An attribute may have one mode, more than one mode (multi-modal), or no mode at all.

```
X.fillna(X['Age'].mode()[0], inplace = True)
```

```
# check that all the missing values are filled

X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           400 non-null    object
 1   Age              400 non-null    int64
 2   EstimatedSalary  400 non-null    int64
dtypes: int64(2), object(1)
memory usage: 9.5+ KB
```

```
# handle missing values in 'Purchased' if any

Y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 1 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Purchased  400 non-null    int64
dtypes: int64(1)
memory usage: 3.2 KB
```

Handling categorical attributes

because distances cannot be calculated on categorical attributes

sklearn.preprocessing.LabelEncoder Encodes target labels with value between 0 and n_classes-1.

Label encoding converts the data in machine-readable form, but it assigns a unique number(starting from 0) to each class of data. This may lead to generation of priority issues in the training datasets. A label with a high value may be considered to have high priority than a label having a lower value.

```
# fit the 'Gender' attribute for label encoding
label_encoder = preprocessing.LabelEncoder().fit(X['Gender'])
```

```
# Encode labels for 'Gender' attribute
X['Gender'] = label_encoder.transform(X['Gender'])
```

```
X.sample(5)
```

	Gender	Age	EstimatedSalary
219	1	59	143000
178	1	24	23000
204	0	58	101000
359	1	42	54000
284	1	48	141000

```
# Go for attribute selection as part of preprocessing

# Check correlation of attributes with target variable

X.corrwith(Y.Purchased, method='pearson')
```

```
Gender            -0.042469
Age                0.622454
EstimatedSalary    0.362083
dtype: float64
```

It can be observed that "Gender" has near to zero correlation with "Purchased". Hence, it can be dropped as part of attribute selection process.

drop() is used to delete rows or columns index label or column name.

DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')

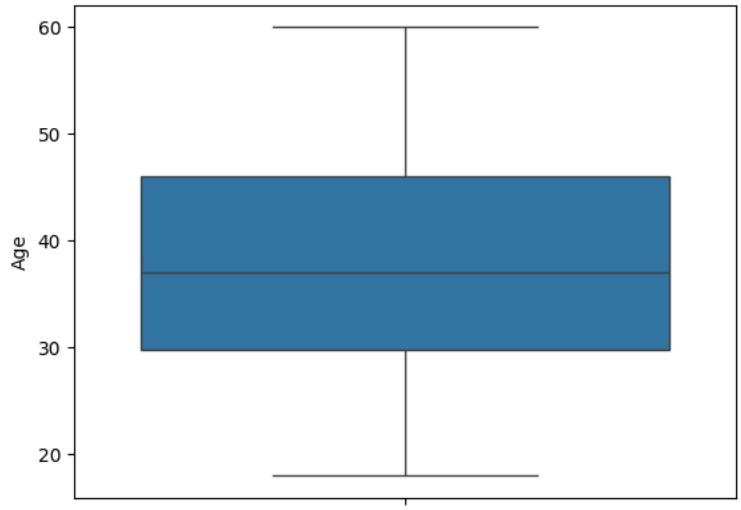
```
labels: String or list of strings referring row or column name.

axis: int or string, 0 'index' for Rows and
             1 'columns' for Columns.

inplace: Makes changes in original Data Frame if True.

Return type: Dataframe with dropped values
```

```
X.drop('Gender', axis = 1, inplace = True)

X.sample(5)
```

	Age	EstimatedSalary
393	60	42000
305	42	54000
336	58	144000
320	52	138000
25	47	20000

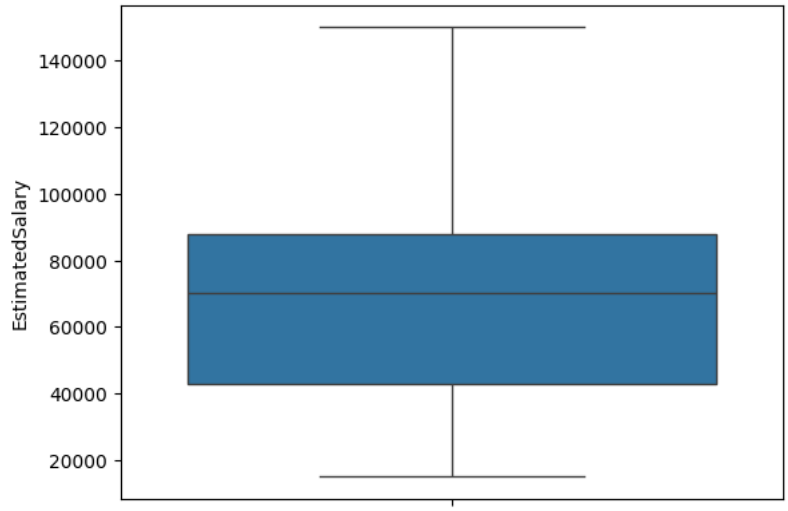
```
# detecting outliers using boxplots

sns.boxplot(X['Age'])
```

```
<Axes: ylabel='Age'>
```




```
sns.boxplot(X['EstimatedSalary'])
```

```
<Axes: ylabel='EstimatedSalary'>
```




```
# Scatter plots or Z-scores can also be used to detect outliers
# if outliers are detected, they must be handled


# Print the description of the dataset
# For each attribute the following will be described -
#          mean, variance, SD, percentiles, min, max


X.describe()
```

	Age	EstimatedSalary
count	400.000000	400.000000
mean	37.655000	69742.500000
std	10.482877	34096.960282
min	18.000000	15000.000000
25%	29.750000	43000.000000
50%	37.000000	70000.000000
75%	46.000000	88000.000000
max	60.000000	150000.000000

Feature Scaling (or Normalization)

Standardization of a dataset is a common requirement for many machine learning estimators:

If a feature has a variance that is orders of magnitude larger that others,

it might dominate the objective function and make the estimator

unable to learn from other features correctly as expected.

MinMaxScaler() class will transform attribute by scaling it to a given range.

It uses formula (x – Xmin) / (Xmax – Xmin)

sklearn.preprocessing.MinMaxScaler(feature_range=(min, max), copy=True)

fit_transform() function fits to data, then transforms it. Fit method will calculate mean and variance of the attribute. Transform method will transform the attribute using the respective mean and variance.

It returns numpy.ndarray

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0, 1))

Fit = scaler.fit(X)

X_ndarray = np.round(Fit.transform(X), 2)

# display normalized values
X_ndarray[0:1]
```

```
array([[0.02, 0.03]])
```

```
X.head(1)
```

	Age	EstimatedSalary
0	19	19000

Next steps: Generate code with X View recommended plots

Min-Max Scaler uses formula = (x – Xmin) / (Xmax – Xmin)

for Age attribute, min = 18, max = 60

Age = 19 will be scaled as = (19 - 18) / (60 - 18) = 0.0238 = 0.02\

train_test_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data.

With this function, you don't need to divide the dataset manually. By default, Sklearn train_test_split will make random partitions for the two subsets.

random_state is basically used for reproducing your problem the same every time it is run.

If you do not use a random_state in train_test_split, every time you make the split you might get a different set of train and test data points and will not help you in debugging in case you get an issue.

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split( X_ndarray, Y,
                                                    test_size=0.25,
                                                   random_state = 35)
```

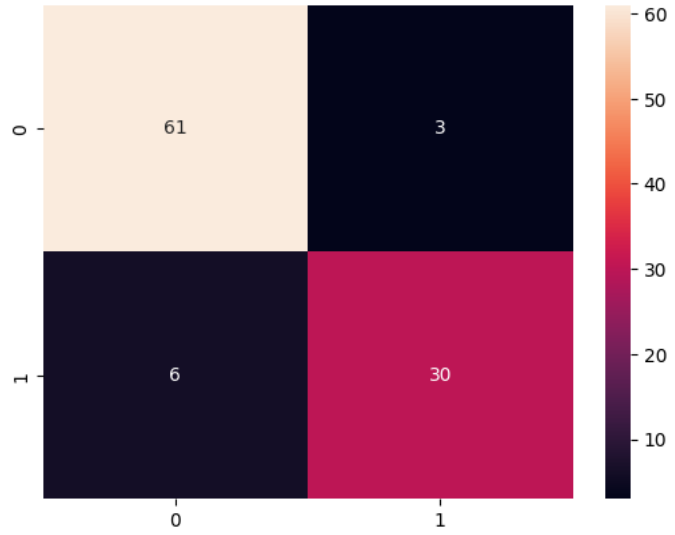
```
# applying KNN with K = 5 and training the model

from sklearn.neighbors import KNeighborsClassifier

# creating an object of KNeighborsClassifier class
knn = KNeighborsClassifier(n_neighbors = 5, metric = 'euclidean')

# train the KNN model
knn.fit(X_train, np.ravel(Y_train))
```

```
KNeighborsClassifier
KNeighborsClassifier(metric='euclidean')
```

```
# testing the model for accuracy of its predictions

# predict() function will make predictions on test dataset

Y_pred = knn.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix , accuracy_score

# make a confusion matrix
cm = confusion_matrix(Y_test, Y_pred)

# display confusion matrix as a heatmap
sns.heatmap(cm, annot = True)

# compute and display the the accuracy of the KNN model
print('The % of Accuracy is :', accuracy_score(Y_test , Y_pred)*100)
```

```
The % of Accuracy is : 91.0
```




```
# use KNN model to make predictions

# reading new customer's data from the .csv file and loading into Dataframe

new_data = pd.read_csv('gdrive/My Drive/ml/New_Data.csv' )

new_data.head()
```

	User ID	Gender	Age	EstimatedSalary
0	453	Female	29	40000

```
new_data.drop(['User ID', 'Gender'], axis = 1, inplace = True)

new_data
```

	Age	EstimatedSalary
0	29	40000

```
# For 'Age' attribute, min = 18, max = 60
# For 'EstimatedSalary' attribute, min = 15000, max = 150000

# performing MinMax Scaling on new data

new_data_ndarray = np.round(Fit.transform(new_data), 2)

# display normalized values
new_data_ndarray[0:1]
```

```
array([[0.26, 0.19]])
```