CSE 362 ASSIGNMENT 2 REPORT

```
dataset = pd.read_csv("purchase_logs.csv")
```

This is reading the purchase_log.csv file.

```
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:, 3].values
```

There are 4 columns in the dataset. X pulls the first 3 columns, and Y pulls the data from the 4th column.

```
labelEncoder_gender =  LabelEncoder()
X[:,0] = labelEncoder_gender.fit_transform(X[:,0])
```

Male and female values are converted to 0 and 1 for easier handling.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

This is a very key point for the whole code. It explains the whole purpose of the code. The code splits some of the data in purchase_logs.csv for training the KNN algorithm and the rest for testing based on this training. The X part is for training, the Y part is for testing the expected result. Finding the 4th column values does the code guessing. The test_size parameter in train_test_split determines how much of the content will be reserved for training and how much for testing. The test_size = 0.2 in the example tells that 20% of the content will be reserved for testing. train_test_split splits data according to the random_state value when splitting, which can be said to be arbitrary. For the same dataset, random_state=0 and random_state=1 separate data differently from each other. However, random_state=0 and random_state=1 run for the same code will be the same as parsing random_state=0 and random_state=1 which will be run later. Based on the "Hitchhiker's Guide to the Galaxy" book, it has become customary to use the random_state value of 42 (there is a reference in the book about achieving perfection of 42).

```
sc_X = StandardScaler()
```

We want to normalize the data so that the data can be statistically processed more healty, and the StandardScaler() method will be used to do this.

```
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

X_train = sc_X.fit_transform(X_train) for normalization of X_train values. X_test = sc_X.transform(X_test) section, the transform() method normalizes the X_test data.

There is a reason why only the transform method is used instead of fit_transform. While the fit_transform method normalizes the X_train data, the transform method is used to transfer the coefficients used for the normalization of the data onto the X_test data. If fit_transform was used on X_test as well, there would be inconsistencies since the two sets would have been normalized differently.

```
knn = KNeighborsClassifier(n_neighbors=60).fit(X_train,y_train)
```

This line of code trains the normalized X_train and y_train values over the specified number of neighbors to train the knn algorithm. In this example, the number of neighbors is 30.

```
y_pred=knn.predict(X_test)
```

Based on the trained algorithm, it predicts the y_pred column from the X_test data reserved for testing.

```
cm = metrics.confusion_matrix(y_test, y_pred)
```

Confusion matrix is used to interpret the predictions made on Knn and to examine the relationship between the actual and predicted values.

```
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy score:",accuracy)
precision = metrics.precision_score(y_test, y_pred)
print("Precision score:",precision)
recall = metrics.recall_score(y_test, y_pred)
print("Recall score:",recall)
```

This section calculates and shows Accuracy, Precision, and Recall results on the expected result

with the prediction dataset, respectively.

Accuracy value shows the percentage of correct predictions made when the column produced by the prediction algorithm is compared with the actual answer column.

The Precision value shows how many of these predictions include positive results when correctly made predictions (ignoring incorrect predictions) are examined (the person who made the purchase will be evaluated positively).

The Recal value, on the other hand, shows the ratio of the customers with a positive shopping status to the sum of the cases where the shopping outcome was predicted to be negative but this prediction was incorrect, and the customers with a positive shopping status predicted correctly.

for k= 10

```
arda@arda-Latitude-E5470:~/Downloads/ai$ /bin/python3 /home/arda/Downloads/ai/Project2.py
Confusion matrix:  [[55  3]
 [ 1 21]]
Accuracy score: 0.95
Precision score: 0.875
Recall score: 0.9545454545454546
```

for k= 30

```
arda@arda-Latitude-E5470:~/Downloads/ai$ /bin/python3 /home/arda/Downloads/ai/Project2.py
Confusion matrix:  [[56  2]
 [ 4 18]]
Accuracy score: 0.925
Precision score: 0.9
Recall score: 0.8181818181818182
```

for k= 60

```
arda@arda-Latitude-E5470:~/Downloads/ai$ /bin/python3 /home/arda/Downloads/ai/Project2.py
Confusion matrix:  [[58  0]
 [ 9 13]]
Accuracy score: 0.8875
Precision score: 1.0
Recall score: 0.5909090909090909
```

It is seen that the selected k value accuracy score has decreased. This does not apply to every dataset. In another dataset, k=30 gives better results than k=10 and k= 60. it could be. The number of people analyzed in this project is 400 and when clustering is done over k=30 and k=60, it has to be included in the scope of other clusters. In another dataset, it can give a healthier result when there is more content, and also if possible, when the clusters are further apart and more clearly clustered.

As the FP (false positive) value decreases towards k=60, the Precision score value increases. Precision score = TP / (TP + FP)

As the FN (false negative) value increases towards k=60, the Recall score value decreases. Recall score = TP / (TP + FN).