```python
#Task 1

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import MinMaxScaler, LabelEncoder

# Load data
cc_apps = pd.read_csv("cc_approvals.data", header=None)

# Data Preprocessing
# Handling Missing Values (replace '?' with NaN)
cc_apps.replace('?', np.nan, inplace=True)
cc_apps.fillna(cc_apps.median(), inplace=True)  # Filling missing values with median for num

# Convert non-numeric columns to strings
for column in cc_apps.columns:
    if cc_apps[column].dtype == 'object':
        cc_apps[column] = cc_apps[column].astype(str)

# Label Encoding for non-numeric columns
label_encoders = {}
for column in cc_apps.columns:
    if cc_apps[column].dtype == 'object':
        label_encoders[column] = LabelEncoder()
        cc_apps[column] = label_encoders[column].fit_transform(cc_apps[column])

# Drop features 11 and 13
cc_apps = cc_apps.drop([11, 13], axis=1)

# Split data into features and target variable
X = cc_apps.drop(columns=[2])  # Replace 'target_column' with your actual target column name
y = cc_apps[2]
```

```python
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)


# Find the best k using Grid Search for KNeighborsRegressor
param_grid = {'n_neighbors': range(1, 21)}  # Trying k values from 1 to 20
knn = KNeighborsRegressor()
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)

# Print the best k value
print("Best K: ", grid_search.best_params_['n_neighbors'])

# Train the final KNN model with the best k value
best_k = grid_search.best_params_['n_neighbors']
knn = KNeighborsRegressor(n_neighbors=best_k)
knn.fit(X_train_scaled, y_train)


test_accuracy = knn.score(X_test, y_test)

print("Accuracy of the best KNN model: ", test_accuracy)
```

```
C:\Users\Lut Lat Aung\AppData\Local\Temp\ipykernel_29180\863855895.py:17: FutureWarning: Drc
ame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise
mns before calling the reduction.
  cc_apps.fillna(cc_apps.median(), inplace=True)  # Filling missing values with median for n

Best K:  18
Accuracy of the best KNN model:  0.4373990252463124 7
```

```python
import pandas as pd
cc_apps = pd.read_csv("cc_approvals.data", header = None)
print(cc_apps.head())
print(cc_apps.describe())
print('\n')
print(cc_apps.info())
print('\n')
cc_apps.tail(17)
cc_apps.fillna(cc_apps.mean(), inplace=True)
print(cc_apps.isnull().sum())
import numpy as np
cc_apps = cc_apps.replace('?', np.NaN)
for col in cc_apps.columns:

    if cc_apps[col].dtypes == 'object':

        cc_apps = cc_apps.fillna(cc_apps[col].value_counts().index[0])
print(cc_apps.isnull().sum())
print(cc_apps.describe())
print('\n')
print(cc_apps.info())
print('\n')
cc_apps.tail(17)
cc_apps = pd.get_dummies(cc_apps)
print(cc_apps)

from sklearn.preprocessing import MinMaxScaler

X_train, y_train = cc_apps.iloc[:, :-1].values, cc_apps.iloc[:, [-1]].values
X_test, y_test = cc_apps.iloc[:, :-1].values, cc_apps.iloc[:, [-1]].values
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX_train = scaler.fit_transform(X_train)
rescaledX_test = scaler.transform(X_test)

from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(rescaledX_train,y_train)
```

Assessing classification performance

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	False Negative	True Positive

- Accuracy:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

Precision

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	**False Positive**
Actual: Fraudulent	False Negative	**True Positive**

- Precision

$$\frac{true\ positives}{true\ positives + false\ positives}$$

Usually, the class of interest is called the positive class. As we aim to detect fraud, the positive class is an illegitimate transaction. So why is the confusion matrix important? There are other important metrics we can calculate from the confusion matrix. Precision is the number of true positives divided by the sum of all positive predictions. It is also called the positive predictive value. In our case, this is the number of correctly labeled fraudulent transactions divided by the total number of transactions classified as fraudulent. High precision means having a lower false positive rate.

Recall

	Predicted: Legitimate	Predicted: Fraudulent
Actual: Legitimate	True Negative	False Positive
Actual: Fraudulent	**False Negative**	**True Positive**

- Recall

$$\frac{true\ positives}{true\ positives + false\ negatives}$$

- High recall = lower false negative rate
- High recall: Predicted most fraudulent transactions correctly

Recall is the number of true positives divided by the sum of true positives and false negatives. This is also called sensitivity. High recall reflects a lower false negative rate. For our classifier, it means predicting most fraudulent transactions correctly.

F1 score

- F1 Score: $2 * \frac{precision * recall}{precision + recall}$

The F1-score is the harmonic mean of precision and recall. This metric gives equal weight to precision and recall, therefore it factors in both the number of errors made by the model and the type of errors. The F1 score favors models with similar precision and recall, and is a useful metric if we are seeking a model which performs reasonably well across both metrics.

```python
from sklearn.model_selection import train_test_split

print(cc_apps.corr())
cc_apps_train, cc_apps_test = train_test_split(cc_apps, test_size=0.33, random_state=42)

from sklearn.metrics import confusion_matrix

y_pred = logreg.predict(rescaledX_test)
print("Accuracy of logistic regression classifier: ", logreg.score(rescaledX_test,y_test))
confusion_matrix(y_test,y_pred)

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

for k in range(1, 10):
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'k = {k}: Accuracy = {accuracy:.2f}')
```