Q1:
Data cleaning: Read the "tranrecords.cvs" file and handle missing values. Depending on the given data format, handle missing entries using an appropriate method (such as filling in the mean, median, or removing missing values).

Exploratory data analysis (EDA): Analyze data and explore the distribution and correlation of different feature columns. Use visualization tools and statistical methods to understand the characteristics and structure of your data.

Feature engineering: Based on the understanding of the data and the results of EDA, select feature columns that are useful for predicting fraudulent transactions. Feature extraction, transformation, or combination may be required to improve model performance.

Model selection and training: Select an appropriate machine learning model for fraudulent transaction prediction. Common models include logistic regression, support vector machines, decision trees, random forests, neural networks, etc. Based on the characteristics of the data and the needs of the problem, select the most appropriate model and use the training data to train the model.

Data preparation: Divide the data into training set and test set. Typically a large portion of the data is used for training and a small portion is used to evaluate the model's performance. Ensure that the partitioning of data is random and maintain the distribution characteristics of the data.

Model training and testing: Use the training set to train the selected model, and use the test set to evaluate the model. Tune and improve the model based on the model's performance indicators (such as accuracy, precision, recall, etc.).

System performance evaluation: Evaluate the overall performance of the system, including model accuracy and efficiency. Methods such as cross-validation, confusion matrix, and ROC curve can be used to evaluate the performance of the model.

Results Interpretation and Deployment: Interpret the model's predictions and make further interpretations and adjustments as needed. Finally, the trained model is deployed into practical applications for real-time prediction and monitoring of fraudulent transactions.

Q2:
```
import gym
import numpy as np

# 创建CartPole环境
env = gym.make('CartPole-v0')

# 设置随机种子
np.random.seed(0)

# 初始化Q表
Q = np.zeros((env.observation_space.n, env.action_space.n))
```

```
# 设置超参数
num_episodes = 1000
max_steps = 200
learning_rate = 0.1
discount_factor = 0.99
exploration_rate = 0.1

# Q-learning算法
for episode in range(num_episodes):
    state = env.reset()
    total_reward = 0

    for step in range(max_steps):
        # 选择动作
        if np.random.uniform(0, 1) < exploration_rate:
            action = env.action_space.sample()  # 随机选择动作
        else:
            action = np.argmax(Q[state, :])  # 根据Q值选择动作

        # 执行动作
        next_state, reward, done, _ = env.step(action)

        # 更新Q值
        Q[state, action] += learning_rate * (reward + discount_factor * np.max(Q[next_state, :]) -
Q[state, action])

        total_reward += reward
        state = next_state

        if done:
            break

    # 输出每个回合的总奖励
    print("Episode:", episode, "Total Reward:", total_reward)

# 使用学习到的Q表玩游戏
state = env.reset()
done = False
total_reward = 0

while not done:
    action = np.argmax(Q[state, :])
    state, reward, done, _ = env.step(action)
    total_reward += reward

# 输出最终的总奖励
print("Final Total Reward:", total_reward)
```

Q3:
a:
y = 4.5*6

b:
i)
The relationship between x and y is such that y = -x. This means that for every element in the sequence x, the corresponding element in the sequence y is its negation. This relationship holds for all elements in the sequence, indicating that there is a close relationship between x and y.

ii)

Q4:

```
import datetime
import os
import flair
from flair.models import TextClassifier
from flair.data import Sentence

# Load the sentiment analysis model
flair_sentiment_model = TextClassifier.load("en-sentiment")

# Function to get the average sentiment score of a list of sentences
def get_average_sentiment_score(sentences):
    total_score = 0
    for sentence in sentences:
        flair_sentiment_model.predict(sentence)
        total_score += sentence.labels[0].score
    if len(sentences) > 0:
        return total_score / len(sentences)
    else:
        return 0

# Load the Bearer Token from the file
with open('BearerToken.txt', 'r') as file:
    bearer_token = file.readline().strip()

# Define the date range (May 7, 2023, to May 16, 2023)
start_date = datetime.datetime(2023, 5, 7)
end_date = datetime.datetime(2023, 5, 16)
date_range = [start_date + datetime.timedelta(days=x) for x in range((end_date -
start_date).days + 1)]

# Analyze tweets for each day in the date range
```

```
for date in date_range:
    formatted_date = date.strftime("%Y-%m-%d")
    query = f'Lenovo OR Dell since:{formatted_date} until:{formatted_date}'
    command = f'curl -X GET
"https://api.twitter.com/2/tweets/search/recent?query={query}&tweet.fields=created_at" -H
"Authorization: Bearer {bearer_token}"'
    response = os.popen(command).read()
    tweets = response.split('\n')
    sentiment_scores = []
    for tweet in tweets:
        if tweet != "":
            sentence = Sentence(tweet)
            sentiment_scores.append(sentence)
    average_sentiment_score = get_average_sentiment_score(sentiment_scores)
    print(f"Date: {formatted_date}, Average Sentiment Score: {average_sentiment_score}")
```