Exercise 1

Task 1

This application uses frame differencing and background subtraction techniques to detect and label cars on the 'Main Street'. Frame differencing involves comparing two video frames and identifying any changes in pixels. Background subtraction involves identifying moving objects by comparing the current frame with a reference frame, referred to as the "background frame" in this case.

Within a while loop that runs through the video, the initial frame is first converted to grayscale and then blurred to remove unnecessary details. The difference between the current frame and the background frame is calculated and stored as the delta frame.

To eliminate false positives caused by lighting variations and noise, various transformations are applied to the video image. The author experimented with different techniques and arrived at a custom threshold called "retvalbin", which improved the accuracy of the contour tags and reduced jitteriness.

```
# Smoothening (Noise Reduction & Grey conversion)
gray_frame=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
blur_frame=cv2.GaussianBlur(gray_frame,(25,25),0)

# 1st image = baseline image
if initial_frame is None:
    initial_frame = blur_frame
    continue

# Finding difference between baseline & frame images
delta_frame=cv2.absdiff(initial_frame,blur_frame)

# The difference between the initial frame and the current frame (the delta_frame) is transformed into a binary imag
# When the pixel value is greater than 18,
# The pixel will be rendered the colour white, otherwise, it is rendered the colour black
# threshold_frame=cv2.threshold(delta_frame,18,255, cv2.THRESH_BINARY)[1]

# The kernel to applies to the morphology
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
closing = cv2.morphologyEx(delta_frame, cv2.MORPH_CLOSE, kernel)
opening = cv2.morphologyEx(closing, cv2.MORPH_OPEN, kernel)
dilation = cv2.dilate(opening, kernel)
retvalbin = cv2.threshold(dilation, 12, 255, cv2.THRESH_BINARY)[1]
```

After applying the final transformation, we use the cv2 library to detect contours and draw bounding rectangles around objects that meet a certain area threshold (>2500). This helps to avoid tagging bicycles and pedestrians. To ensure that we only tag cars on the 'Main Street', we also limit the tags to the lower half of the frame. This restriction has proven to be quite accurate.

```
# cv2.findContours() method allows us to identify the contours in the image
(contours,_)=cv2.findContours(retvalbin,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

# Gets height of picture frame
height = 0
if video.isOpened():
    height = video.get(cv2.CAP_PROP_FRAME_HEIGHT)

# This loops over each contour
# Paints them on original image
for c in contours:
    # contourArea() will filters out the small contours
    if cv2.contourArea(c) < 2500:
        continue
    (x, y, w, h)=cv2.boundingRect(c)
    # If a car on main street is near the bottom half of the video frame, paint the contour over it
    if y > height / 2:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 1)

# We can use the different frames generated here.
cv2.imshow('Video', frame)
```

Task 2

Expanding on Task 1, which utilizes the same transformations and tags, I developed a class named "box" that serves as a collision object. The box class verifies whether an object has intersected with it, and then increments a counter.

```
import cv2

# Point of collision will be represented by 'Box' class
class Box:
    def __init__(self, start_point, width_height):
        self.start_point = start_point
        self.end_point = (start_point[0] + width_height[0], start_point[1] + width_height[1])
        self.counter = 0
        self.frame_countdown = 0

    def overlap(self, start_point, end_point):
        if self.start_point[0] >= end_point[0] or self.end_point[0] <= start_point[0] or \
                self.start_point[1] >= end_point[1] or self.end_point[1] <= start_point[1]:
            return False
        else:
            return True
```

A box is first created as seen from the Boxes.Append object. The cv2 library's Image Moments functionality is utilized to compute various characteristics, including the centroid of an object. This proved to be more advantageous than simply checking if the entire contour intersected the box. Using cv2 moments, a dictionary of all moment values is generated, and we extract the centroid values (cX and cY). During the contour loop where cars are tagged, we check if the centroid of each contour overlaps with the box. If an overlap is detected, the counter is incremented.

```
    cX = int(M["m10"] / M["m00"])
    cY = int(M["m01"] / M["m00"])
    (x, y, w, h)=cv2.boundingRect(c)
    # If a car on main street is near the bottom half of the video frame, paint the contour over it
    if y > height / 2:
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 1)
        cv2.circle(frame, (cX, cY), 7, (255, 255, 255), -1)


    # Constructing of texts strings
        cars = 0
        # Going through each box
        for box in boxes:
            box.frame_countdown -= 1
            if box.overlap((cX, cY), (x + w, y + h)):
                if box.frame_countdown <= 0:
                    box.counter += 1
                box.frame_countdown = 20
            cars += box.counter

text = "Cars: " + str(cars)
# Setting up text strings being built
cv2.putText(frame, text, (10, 20), cv2.FONT_HERSHEY_PLAIN, 2, (0, 255, 0), 2)

for box in boxes:
    cv2.rectangle(frame, box.start_point, box.end_point, (255, 255, 255), 2)

# We can use the different frames generated here.
cv2.imshow('Video', frame)
```

Upon completion of the video, the cell output must display the total number of cars detected and the number of cars per minute. This is accomplished by utilizing the car counter and computing the cars per minute value by dividing the total number of cars by the video duration.

```
# Getting frames, fps and duration of video
frames = video.get(cv2.CAP_PROP_FRAME_COUNT)
fps = int(video.get(cv2.CAP_PROP_FPS))
seconds = int(frames / fps)
minutes = seconds / 60
```

Output from the videos are as follows:

	Total number of cars	**Cars per minute**
Traffic_Laramie_1.mp4	9	3.05
Traffic_Laramie_2.mp4	4	2.29