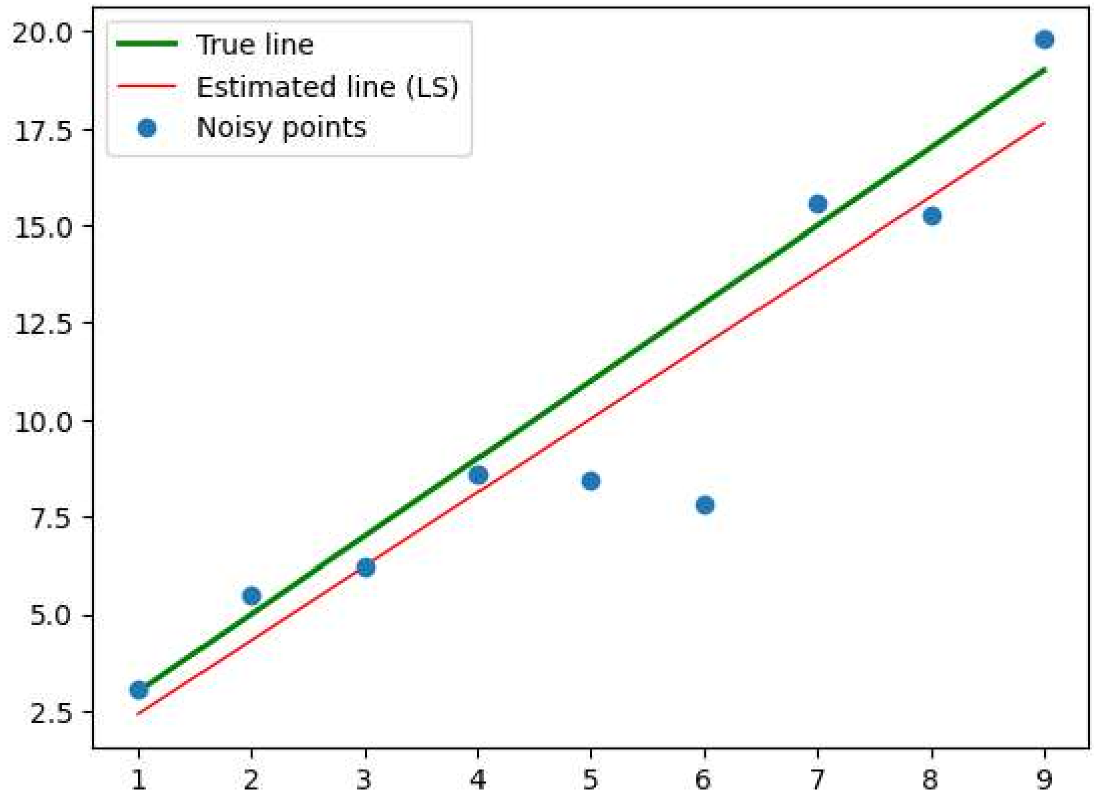
In [ ]:

```
#1
import numpy as np
import matplotlib.pyplot as plt

# Genrating the true line y = m*x + c
m = 2 # gradient
c = 1 # intercept
x = np.arange(1,10, 1)
np.random.seed(45)
n = 2.*np.random.randn(len(x))
o = np.zeros(x.shape)

#o[-1] = 20

y = m*x + c + n + o


X = np.concatenate([x.reshape(len(x),1), np.ones((len(x), 1))], axis=1)
B = np.linalg.pinv(X.T @ X) @ X.T @ y
mstar = B[0]
cstar = B[1]

plt.plot([x[0], x[-1]], [m*x[0] + c, m*x[-1] + c], color='g', linewidth=2, label
plt.plot([x[0], x[-1]], [mstar*x[0] + cstar, mstar*x[-1] + cstar], color='r', li
plt.plot(x,y, 'o', label='Noisy points')
plt.legend(loc='best')
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x1e544fdced0>
```

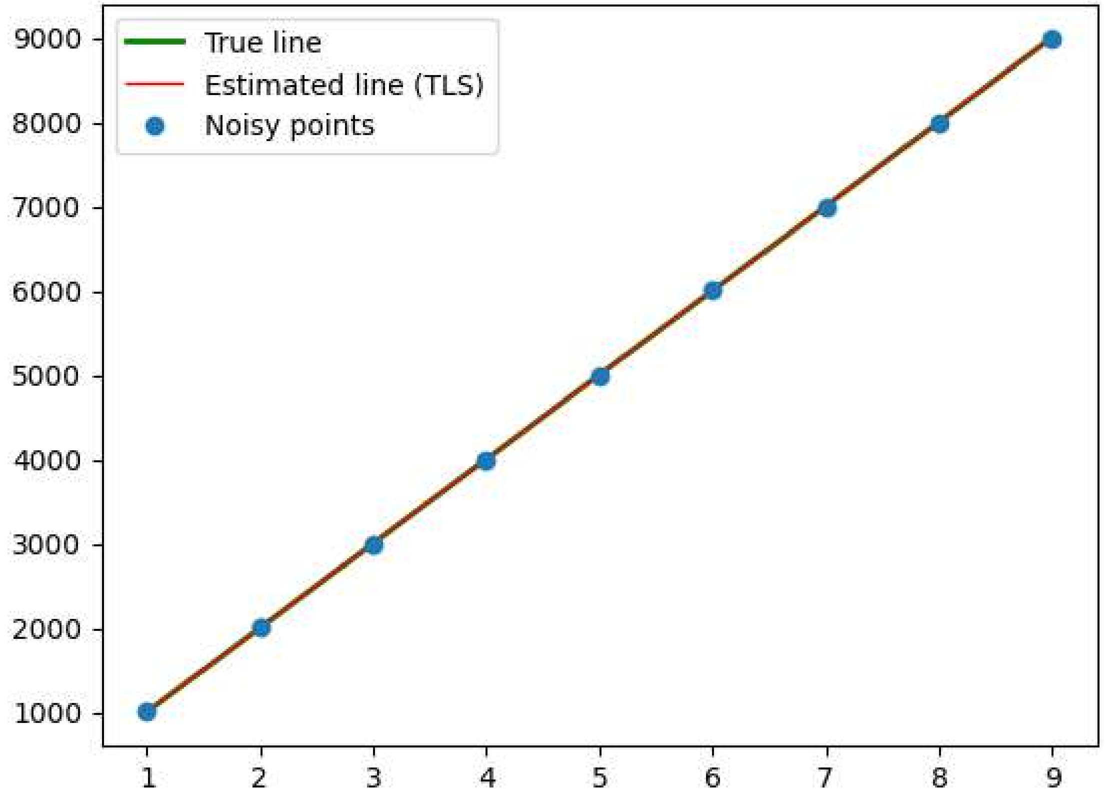


In [ ]:

```
#2
import numpy as np
import matplotlib.pyplot as plt
```

```
# Genrating the true line y = m*x + c
m = 1000 # gradient
c = 1 # intercept
x = np.arange(1,10, 1)
np.random.seed(45)
noise = np.random.randn(len(x))
o = np.zeros(x.shape)
#o[-2] = 28
o[-4] = 18
y = m*x + c + noise + o

n = len(x)

u11 = np.sum((x - np.mean(x))**2)
u12 = np.sum((x - np.mean(x))*(y - np.mean(y)))
u21 = u12
u22 = np.sum((y - np.mean(y))**2)
U = np.array([[u11, u12], [u21, u22]])
w, v = np.linalg.eig(U)
smallest_eigenvector = v[:, np.argmin(w)]
a = smallest_eigenvector[0]
b = smallest_eigenvector[1]
d = a*np.mean(x) + b*np.mean(y)
mstar = -a/b
cstar = d/b



plt.plot([x[0], x[-1]], [m*x[0] + c, m*x[-1] + c], color='g', linewidth=2, label
plt.plot([x[0], x[-1]], [mstar*x[0] + cstar, mstar*x[-1] + cstar], color='r', li
plt.plot(x,y, 'o', label='Noisy points')
plt.legend(loc='best')


#for large values of m (ex-1000) true line and estimated line overlap
```

```
Out[ ]: <matplotlib.legend.Legend at 0x1e5452b4b50>
```

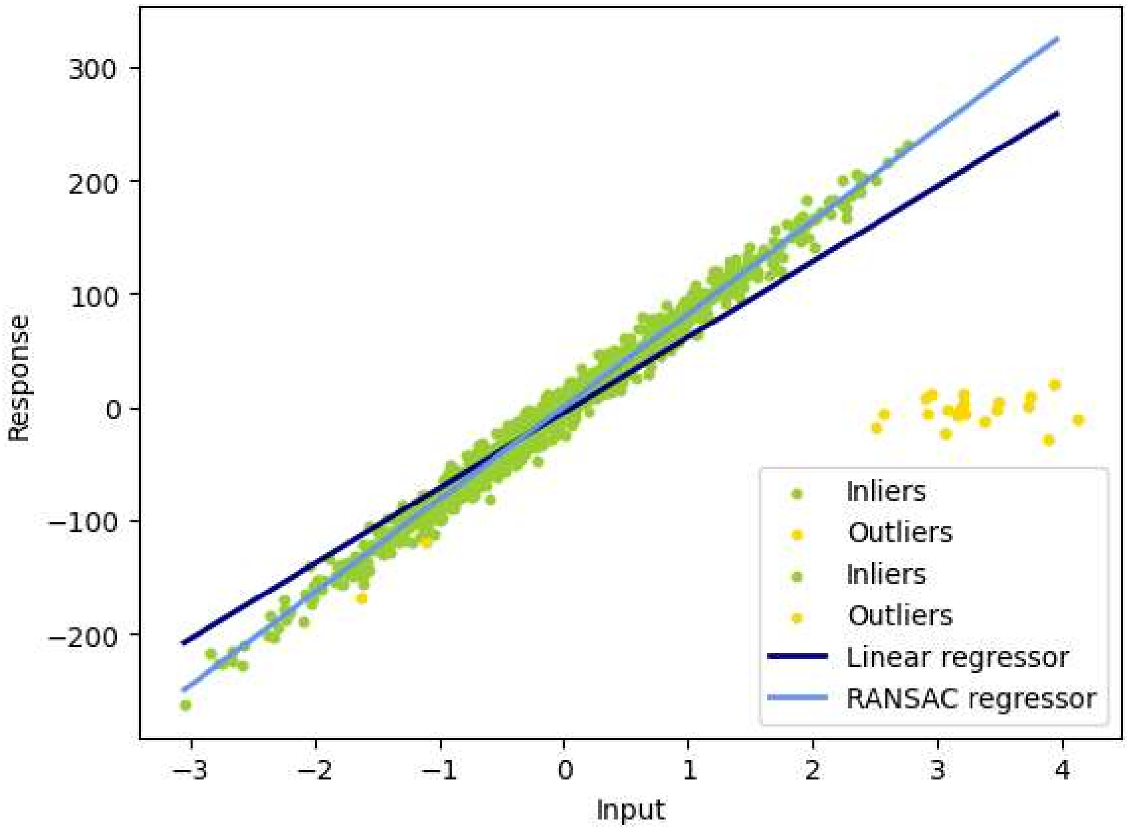


In [ ]:

```
#3
import numpy as np
from matplotlib import pyplot as plt

from sklearn import linear_model, datasets


n_samples = 1000
n_outliers = 20


X, y, coef = datasets.make_regression(
    n_samples=n_samples,
    n_features=1,
    n_informative=1,
    noise=10,
    coef=True,
    random_state=0,
)

# Add outlier data
np.random.seed(0)
X[:n_outliers] = 3 + 0.5 * np.random.normal(size=(n_outliers, 1))
y[:n_outliers] = -3 + 10 * np.random.normal(size=n_outliers)


plt.scatter(
    X[:n_samples], y[:n_samples], color="yellowgreen", marker=".", label="Inlier
)
plt.scatter(
    X[:n_outliers], y[:n_outliers], color="gold", marker=".", label="Outliers"
)

lr = linear_model.LinearRegression()
```

```
lr.fit(X, y)


ransac = linear_model.RANSACRegressor()
ransac.fit(X, y)
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

line_X = np.arange(X.min(), X.max())[:, np.newaxis]
line_y = lr.predict(line_X)
line_y_ransac = ransac.predict(line_X)
print
print("Estimated coefficients (true, linear regression, RANSAC):")
print(coef, lr.coef_, ransac.estimator_.coef_)

lw = 2
plt.scatter(
    X[inlier_mask], y[inlier_mask], color="yellowgreen", marker=".", label="Inli
)
plt.scatter(
    X[outlier_mask], y[outlier_mask], color="gold", marker=".", label="Outliers"
)
plt.plot(line_X, line_y, color="navy", linewidth=lw, label="Linear regressor")
plt.plot(
    line_X,
    line_y_ransac,
    color="cornflowerblue",
    linewidth=lw,
    label="RANSAC regressor",
)
plt.legend(loc="lower right")
plt.xlabel("Input")
plt.ylabel("Response")
plt.show()
```

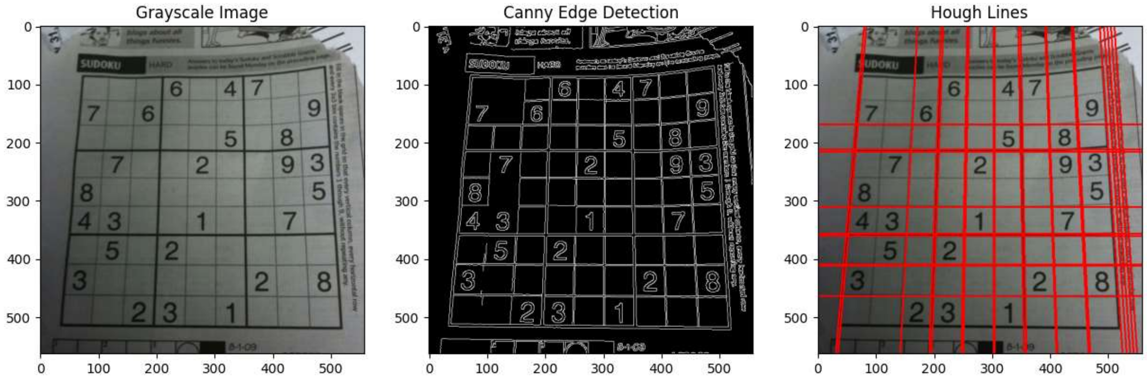
```
Estimated coefficients (true, linear regression, RANSAC):
82.1903908407869 [66.61772415] [82.00627125]
```



In [ ]:

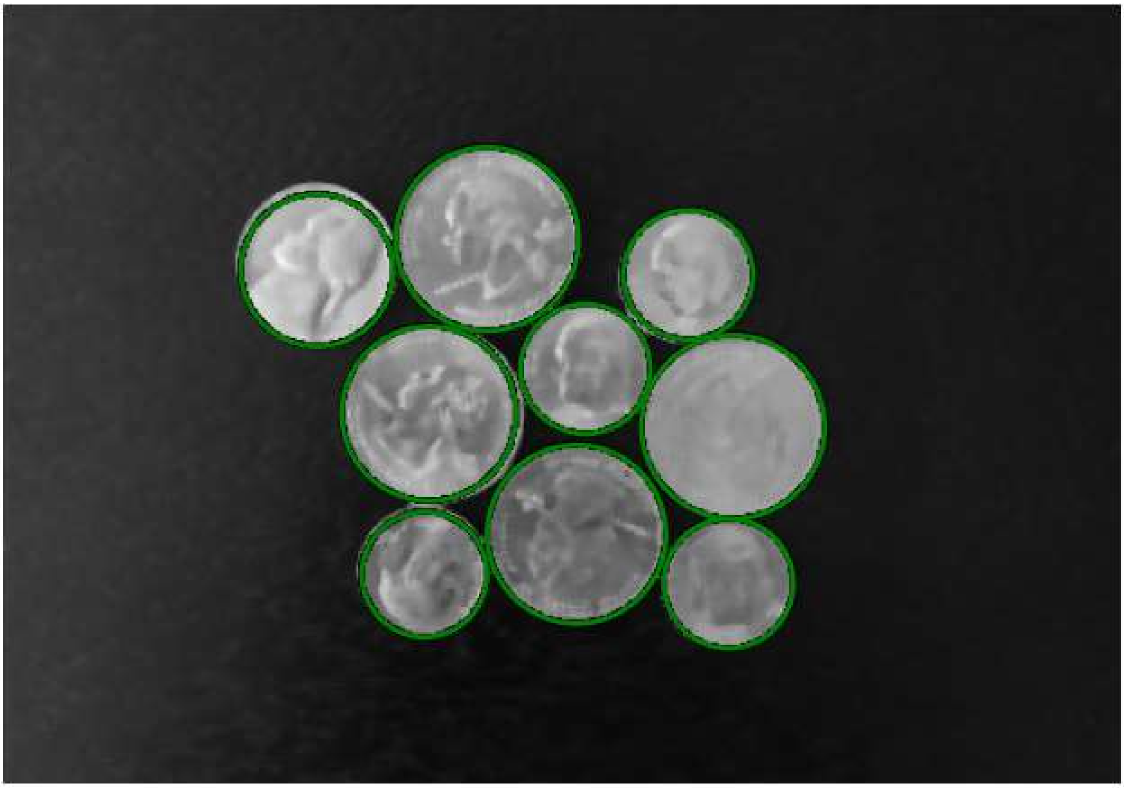
```python
#4
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

im = cv.imread(r'sudoku.png', cv.IMREAD_COLOR)
assert im is not None

gray = cv.cvtColor(im, cv.COLOR_BGR2BGRA)
edges = cv.Canny(gray, 50, 150, apertureSize=3)
lines = cv.HoughLines(edges, 1, np.pi/180, 200)

for line in lines:
    rho, theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv.line(im, (x1, y1), (x2, y2), (0, 0, 255), 2)

# Plot the three images using matplotlib
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].imshow(cv.cvtColor(gray, cv.COLOR_BGR2RGB))
axs[0].set_title('Grayscale Image')
axs[1].imshow(edges, cmap='gray')
axs[1].set_title('Canny Edge Detection')
axs[2].imshow(cv.cvtColor(im, cv.COLOR_BGR2RGB))
axs[2].set_title('Hough Lines')
```

```
plt.show()
```



In [ ]:

```
#5
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread('coins.jpg', cv.IMREAD_GRAYSCALE)
img = cv.medianBlur(img,5)

circles = cv.HoughCircles(img, cv.HOUGH_GRADIENT, 1, 20,
                          param1=180, param2=50, minRadius=0, maxRadius=0)

if circles is not None:
    circles = np.round(circles[0, :]).astype("int")
    for (x, y, r) in circles:
        cv.circle(img, (x, y), r, (0, 255, 0), 2)

fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
if circles is not None:
    for (x, y, r) in circles:
        ax.add_patch(plt.Circle((x, y), r, color='g', fill=False, linewidth=2))
plt.axis('off')
plt.show()
```



In [ ]:

```
#6
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('pic1.png', cv2.IMREAD_GRAYSCALE)
template = cv2.imread('templ.png', cv2.IMREAD_GRAYSCALE)

image_grad_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
image_grad_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)

image_grad_mag = np.sqrt(image_grad_x ** 2 + image_grad_y ** 2)
image_grad_dir = np.arctan2(image_grad_y, image_grad_x)
template_grad_x = cv2.Sobel(template, cv2.CV_64F, 1, 0, ksize=3)
template_grad_y = cv2.Sobel(template, cv2.CV_64F, 0, 1, ksize=3)
template_grad_mag = np.sqrt(template_grad_x ** 2 + template_grad_y ** 2)
template_grad_dir = np.arctan2(template_grad_y, template_grad_x)

grad_template = cv2.matchTemplate(image_grad_mag, template_grad_mag, cv2.TM_CCOE
orient_template = cv2.matchTemplate(image_grad_dir, template_grad_dir, cv2.TM_CC

hough_space = np.zeros_like(img)
for y in range(hough_space.shape[0]):
    for x in range(hough_space.shape[1]):
        if grad_template[y, x] > 0.7:
            theta = orient_template[y, x] * 180 / np.pi
            rho = x * np.cos(theta) + y * np.sin(theta)
            rho_idx = int(rho + hough_space.shape[1] / 2)
            theta_idx = int(theta / 2)
            hough_space[rho_idx, theta_idx] += 1

peaks = cv2.HoughPeaks(hough_space, 10, threshold=0.4*np.max(hough_space))

for peak in peaks[0]:
    rho = peak[0] - hough_space.shape[1] / 2
    theta = peak[1] * 2 * np.pi / 180
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a * rho
    y0 = b * rho
    x1 = int(x0 + 1000 * (-b))
    y1 = int(y0 + 1000 * a)
    x2 = int(x0 - 1000 * (-b))
    y2 = int(y0 - 1000 * a)
    cv2.line(img, (x1, y1), (x2, y2), 255, 2)

fig, ax = plt.subplots(2, 2, figsize=(10, 10))
ax[0, 0].imshow(img, cmap='gray')
ax[0, 0].set_title('Image')
ax[0, 1].imshow(template, cmap='gray')
ax[0, 1].set_title('Template')
plot.show()
```

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
Cell In[41], line 19
     16 template_grad_mag = np.sqrt(template_grad_x ** 2 + template_grad_y **
2)
     17 template_grad_dir = np.arctan2(template_grad_y, template_grad_x)
---> 19 grad_template = cv2.matchTemplate(image_grad_mag, template_grad_mag, cv
2.TM_CCOEFF_NORMED)
     20 orient_template = cv2.matchTemplate(image_grad_dir, template_grad_dir,
cv2.TM_CCOEFF_NORMED)
     22 hough_space = np.zeros_like(img)

error: OpenCV(4.7.0) D:\a\opencv-python\opencv-python\opencv\modules\imgproc\sr
c\templmatch.cpp:1164: error: (-215:Assertion failed) (depth == CV_8U || depth
== CV_32F) && type == _templ.type() && _img.dims() <= 2 in function 'cv::matchT
emplate'
```