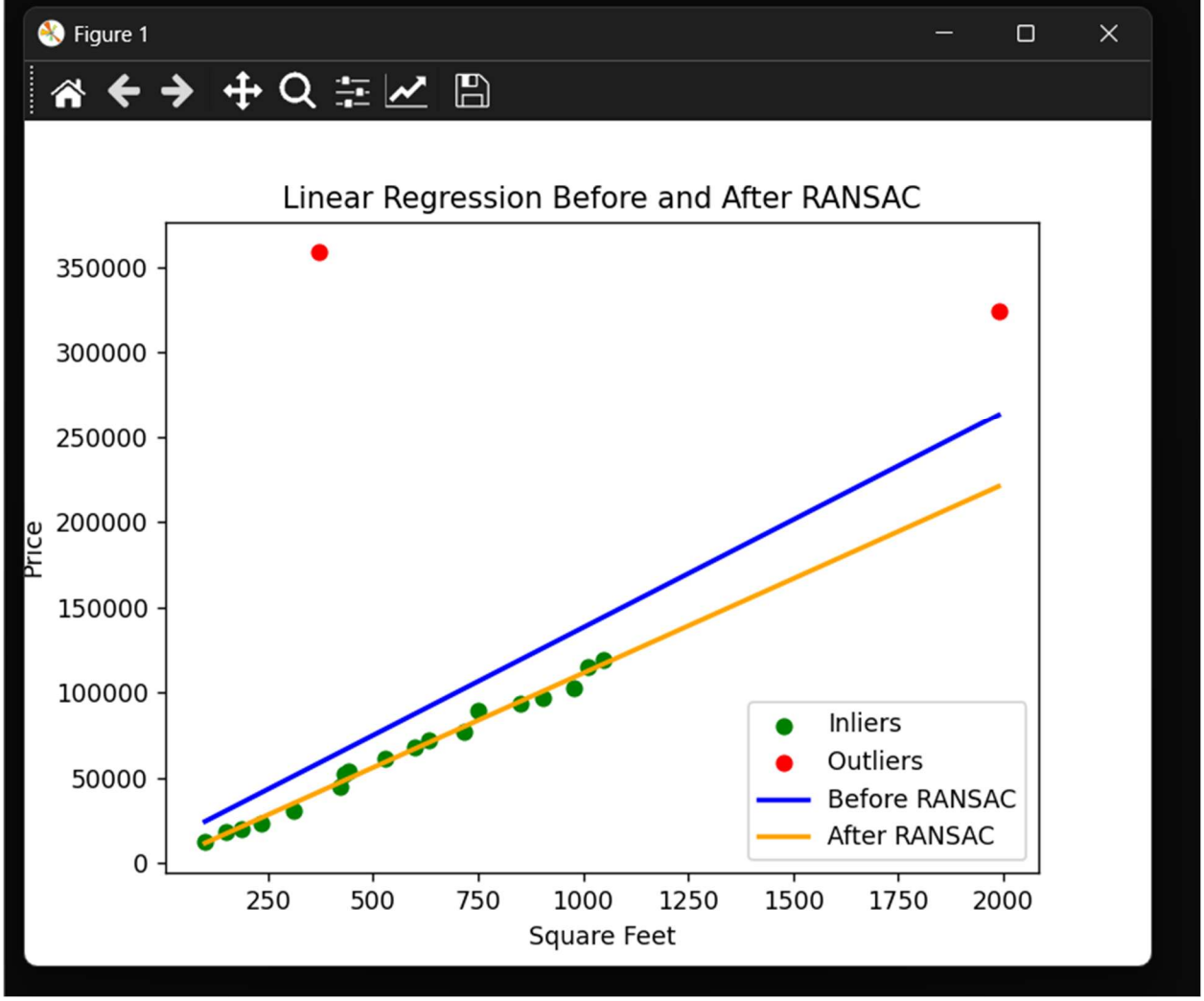
1.

```python
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn import datasets, linear_model

data = {
    'SquareFeet' :[100, 150, 185, 235, 310, 370, 420, 430, 440, 530, 600,
                634, 718, 750, 850, 903, 978, 1010, 1050, 1990],
    'Price' : [12300, 18150, 20100, 23500, 31005,359000, 44359, 52000, 53853,
        61328, 68000, 72300, 77000, 89379, 93200, 97150, 102750, 115358, 119330, 323989]
    }

df = pd.DataFrame(data)

X = np.array(df['SquareFeet']).reshape(-1, 1)
y = np.array(df['Price'])

# Fit line using all data
lr = linear_model.LinearRegression()
lr.fit(X, y)

# Robustly fit linear model with RANSAC algorithm
ransac = linear_model.RANSACRegressor()
ransac.fit(X, y)
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

# Predict data of estimated models
line_X = np.arange(X.min(), X.max())[:, np.newaxis]
line_y = lr.predict(line_X)
line_y_ransac = ransac.predict(line_X)

# Compare estimated coefficients
print(f"Before RANSAC: slope: {lr.coef_}, y-intercept: {lr.intercept_}")
print(f"After RANSAC: slope: {ransac.estimator_.coef_}, y-intercept: {ransac.estimator_.intercept_}")

plt.scatter(X[inlier_mask], y[inlier_mask], color="g", label="Inliers")
plt.scatter(X[outlier_mask], y[outlier_mask], color="r", label="Outliers")
plt.plot(line_X, line_y, color="b", linewidth=2, label="Before RANSAC")
plt.plot(line_X, line_y_ransac, color="orange", linewidth=2, label="After RANSAC")
plt.legend()
plt.xlabel("Square Feet")
plt.ylabel("Price")
plt.title("Linear Regression Before and After RANSAC")
plt.show()
```

C:\WINDOWS\system32\cmd.

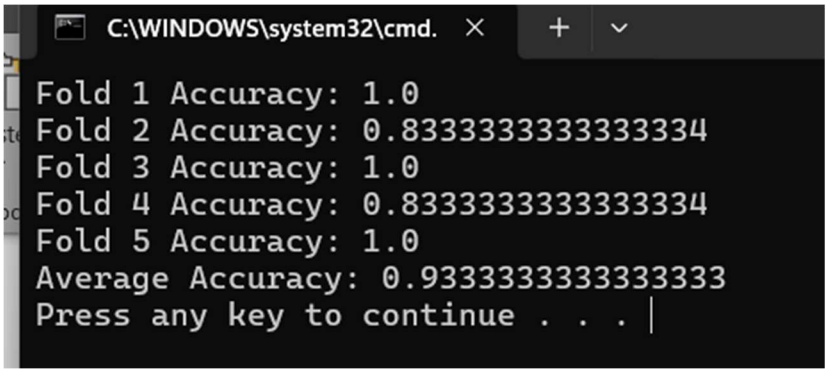
```
Before RANSAC: slope: [126.41285978], y-intercept: 11727.454260912462
After RANSAC: slope: [110.7619806], y-intercept: 610.4963138353487
```



2.

```python

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold, cross_val_score
import math

# Load dataset
names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class']
df = pd.read_csv('iris.data.csv', header=None, names=names)
df['Index'] = [x for x in range(len(df))]

y = np.array(df['class'])
df = df.drop(['class'], axis=1)
X = np.array(df)

np.set_printoptions(precision=2, suppress=True) # Suppress scientific notation


'''
1. we split the dataset into k number of subsets (known as folds)
2.then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model.


In this method, we iterate k times with a different subset reserved for testing purpose each time. The values for k can include three, five, and ten,
with two of the most common being k = 5 and k = 10. In each iteration, one fold is for testing, and the remaining k-1
folds are for training. After testing, you calculate the average of the results. K-fold cross-validation is commonly used
and highly adaptable to a variety of data sets.
'''

# My K-Fold Cross Validation

```

```python
def Myk_fold(n_folds, X, y):
    fold_size = math.floor(len(X) / n_folds)  # Size of each fold
    accuracies = []

    for i in range(n_folds):
        # take this itterations test set (eval)
        test_start = i * fold_size
        test_end = (i + 1) * fold_size
        X_test = X[test_start:test_end]
        y_test = y[test_start:test_end]
        # take the rest of the data to be the train data
        X_train = np.concatenate([X[:test_start], X[test_end:]], axis=0)
        y_train = np.concatenate([y[:test_start], y[test_end:]], axis=0)

        #preform KNN
        knn = KNeighborsClassifier(n_neighbors=9)
        knn.fit(X_train, y_train)
        #get accuracy
        accuracy = knn.score(X_test, y_test)
        accuracies.append(accuracy)
        print(f"Fold {i+1} Accuracy: {accuracy}")

    # Print the average accuracy across all folds
    print(f"Average Accuracy: {np.mean(accuracies)}")


Myk_fold(5, X, y)

```


C:\WINDOWS\system32\cmd.
Fold 1 Accuracy: 1.0
Fold 2 Accuracy: 0.8333333333333334
Fold 3 Accuracy: 1.0
Fold 4 Accuracy: 0.8333333333333334
Fold 5 Accuracy: 1.0
Average Accuracy: 0.9333333333333333
Press any key to continue . . .