P3

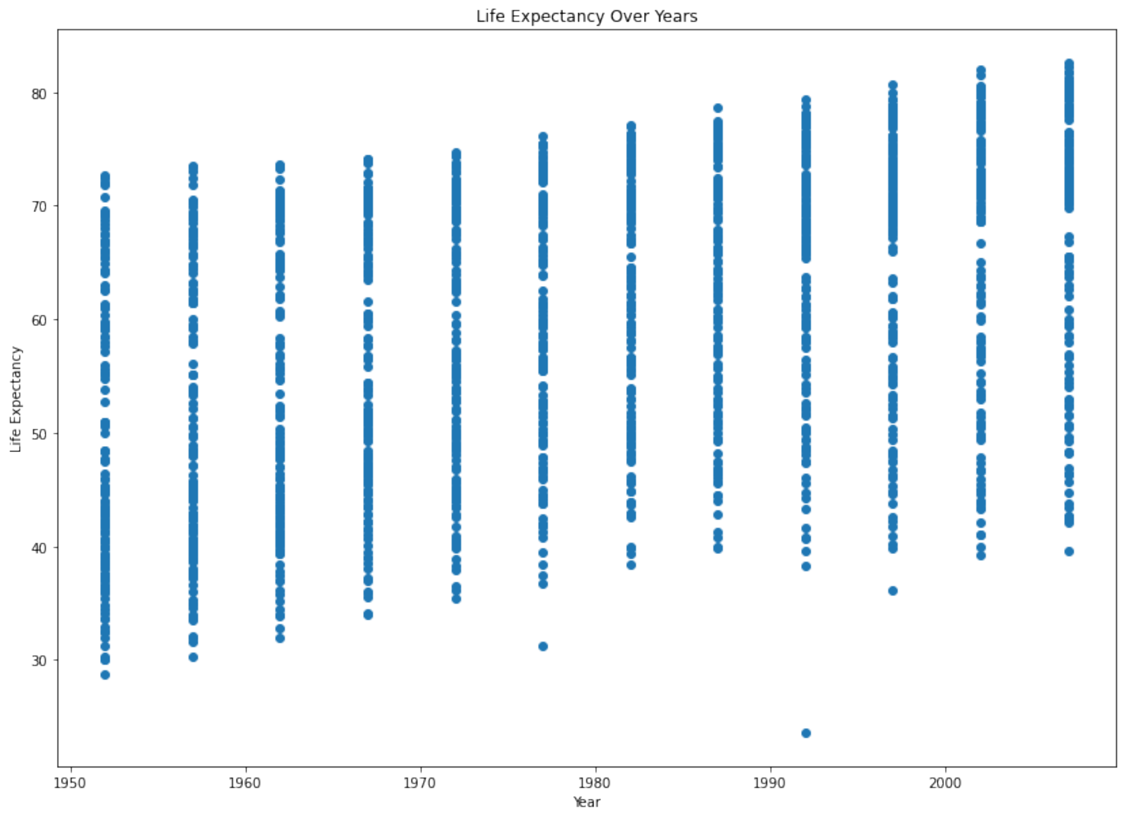
December 2, 2022

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
```

[ ]:
```
data = pd.read_csv('08_gap-every-five-years.tsv', sep='\t')
data
```

[ ]:
```
          country continent  year  lifeExp       pop   gdpPercap
0     Afghanistan      Asia  1952   28.801   8425333  779.445314
1     Afghanistan      Asia  1957   30.332   9240934  820.853030
2     Afghanistan      Asia  1962   31.997  10267083  853.100710
3     Afghanistan      Asia  1967   34.020  11537966  836.197138
4     Afghanistan      Asia  1972   36.088  13079460  739.981106
…             …         …    …       …         …          …
1699     Zimbabwe    Africa  1987   62.351   9216418  706.157306
1700     Zimbabwe    Africa  1992   60.377  10704340  693.420786
1701     Zimbabwe    Africa  1997   46.809  11404948  792.449960
1702     Zimbabwe    Africa  2002   39.989  11926563  672.038623
1703     Zimbabwe    Africa  2007   43.487  12311143  469.709298

[1704 rows x 6 columns]
```

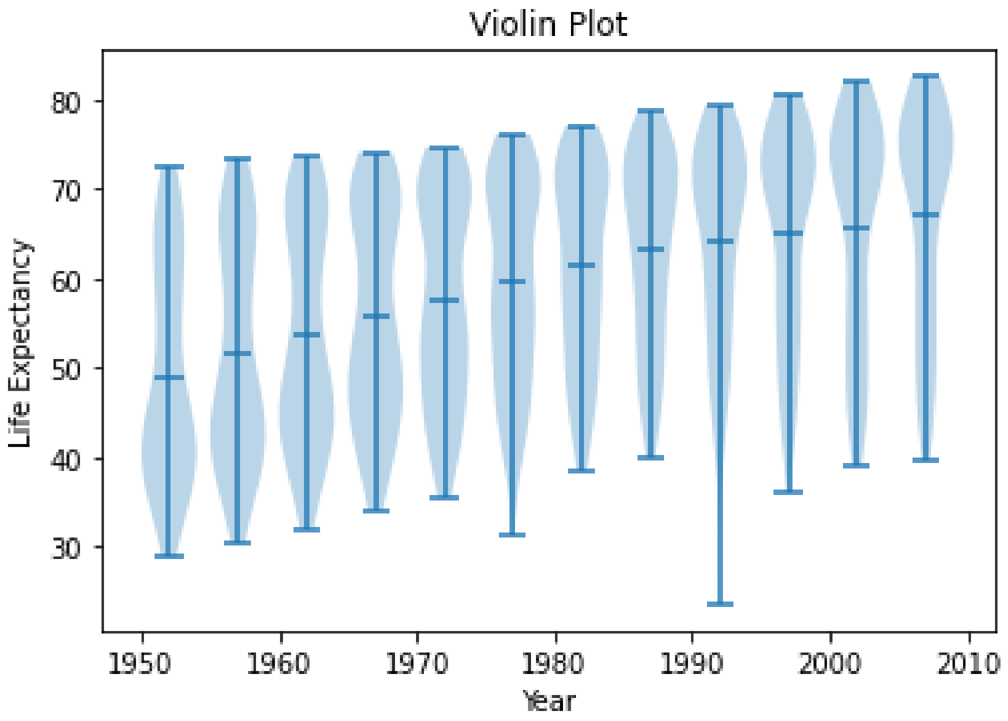
[ ]:
```
## Exercise 1

# Making the scatterplot with life expectancy over time using matplotlib

plt.figure(figsize=(14,10))
plt.title('Life Expectancy Over Years')
plt.xlabel('Year')
plt.ylabel('Life Expectancy')
plt.scatter(data['year'], data['lifeExp'])
plt.show()
```



##Question 1 Looking at the scatter plot, excluding the outliers, there seems to be a linear increase in life expectancy over the years. We can see that the life expectancy increase for each year sampled appears to increase at a constant rate thus the trend is linear.

```
[ ]: # Converting all the life expectancy data for each year into arrays and␣
      ↪appending them into a 2D array
     # so that the data can be used to generate the violin plot in matplotlib

     years_list = data['year'].unique()
     violin_list = []

     for year in years_list:
       filtered = data.loc[data['year'] == year]['lifeExp']
       violin_list.append(filtered)


     fig, ax = plt.subplots()

     ax.violinplot(violin_list,years_list,widths=4,showmeans=True)
     ax.set_xlabel("Year")
     ax.set_ylabel("Life Expectancy")
     ax.set_title("Violin Plot")
```

```
[ ]: Text(0.5, 1.0, 'Violin Plot')
```



##Question 2 Looking at the violin plot, it is apparent that for the first few years, the data is skewed towards lower life expectancy. From 1952 to 1962, we see that there is a greater concentration of data that lies below the mean, thus the data unimodal and not symmetrical.

For 1967 and 1972, the data is somewhat balanced giving the violins a relatively symmetrical appearance. In addition, most of the data in these two years are evenly split between above and below the mean showing the data is possibly bimodal.

For the remaining years from 1977 to 2007, it is apparent that the data is skewed towards the higher life expectancies as most of the data is concentrated above the means. As such, the data is once again unimodal and not symmetrical.

##Question 3 Intuitively I would reject the null hypothesis since it is common knowledge that with the decrease in wars and advancements in technology and society over the years, human life expectancy is the highest it has ever been. Therefore I believe there is a positive correlation between year and life expectancy.

##Question 4 A violin plot of the residuals would look linear.

##Question 5 Based off assumptions of the linear regression model, the violin plot of the residual would be normally distributed around a 0 mean.

```
[ ]: from sklearn import linear_model
```

```
## Exercise 2

# Using SKLearn's linear_model to fit life expectancy and year data into a␣
 ↪linear regression model

mdl = linear_model.LinearRegression()

# LinearRegression().fit() needs a 2D array for first argument so reshaping to␣
 ↪2D array
x = np.array(data['year']).reshape((-1, 1))
y = np.array(data['lifeExp'])
mdl.fit(x, y)
print('Linear Model: f(x) = ax + b where:')
print('a = {}'.format(mdl.coef_))
print('b = {}'.format(mdl.intercept_))
```

```
Linear Model: f(x) = ax + b where:
a = [0.32590383]
b = -585.6521874415448
```

##Question 6 On average we see a life expectancy increase of 0.32589383 years per year.

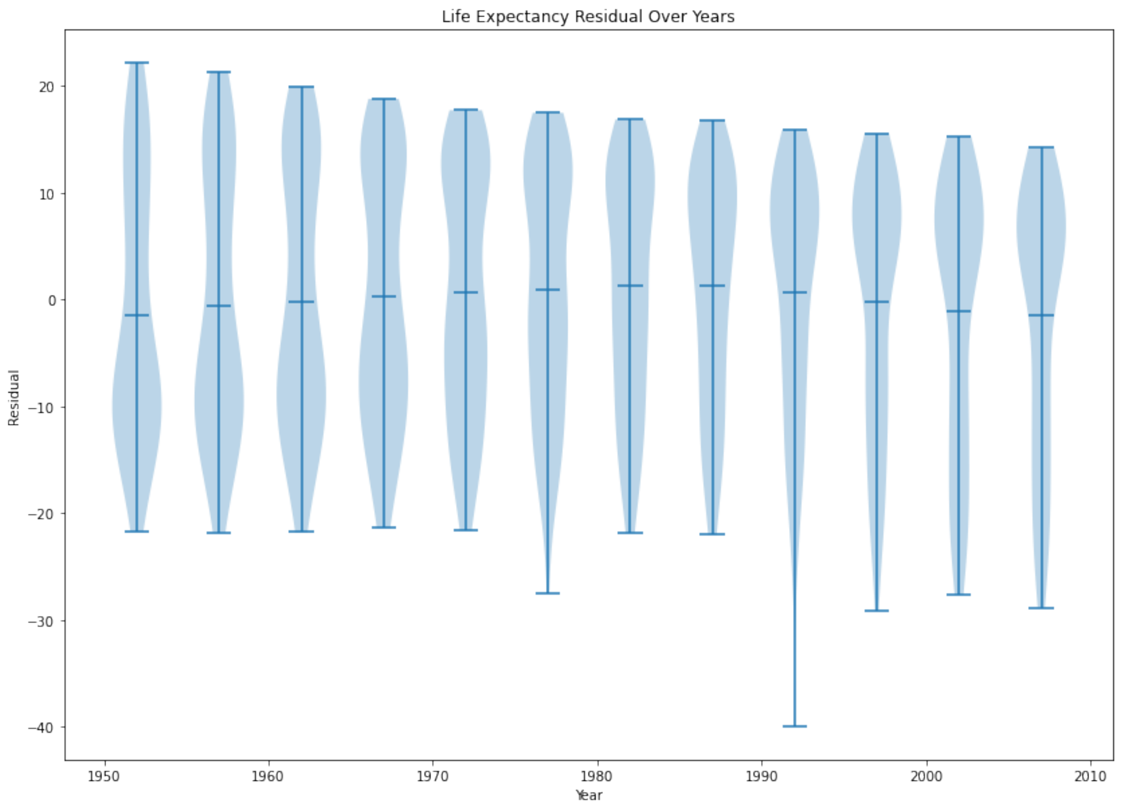
[ ]:
```
# Get summary of linear regression model between life expectancy and year using␣
 ↪statsmodels' Ordinary Least Squares regression

import statsmodels.formula.api as sm
mdl_sum = sm.ols(formula='lifeExp ~ year', data=data).fit()
print(mdl_sum.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lifeExp   R-squared:                       0.190
Model:                            OLS   Adj. R-squared:                  0.189
Method:                 Least Squares   F-statistic:                     398.6
Date:                Wed, 30 Nov 2022   Prob (F-statistic):           7.55e-80
Time:                        23:32:56   Log-Likelihood:                -6597.9
No. Observations:                1704   AIC:                         1.320e+04
Df Residuals:                    1702   BIC:                         1.321e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   -585.6522     32.314    -18.124      0.000    -649.031    -522.273
year           0.3259      0.016     19.965      0.000       0.294       0.358
==============================================================================
Omnibus:                      386.124   Durbin-Watson:                   0.197
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               90.750
```

```
Skew:                          -0.268   Prob(JB):                     1.97e-20
Kurtosis:                       2.004   Cond. No.                     2.27e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.27e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

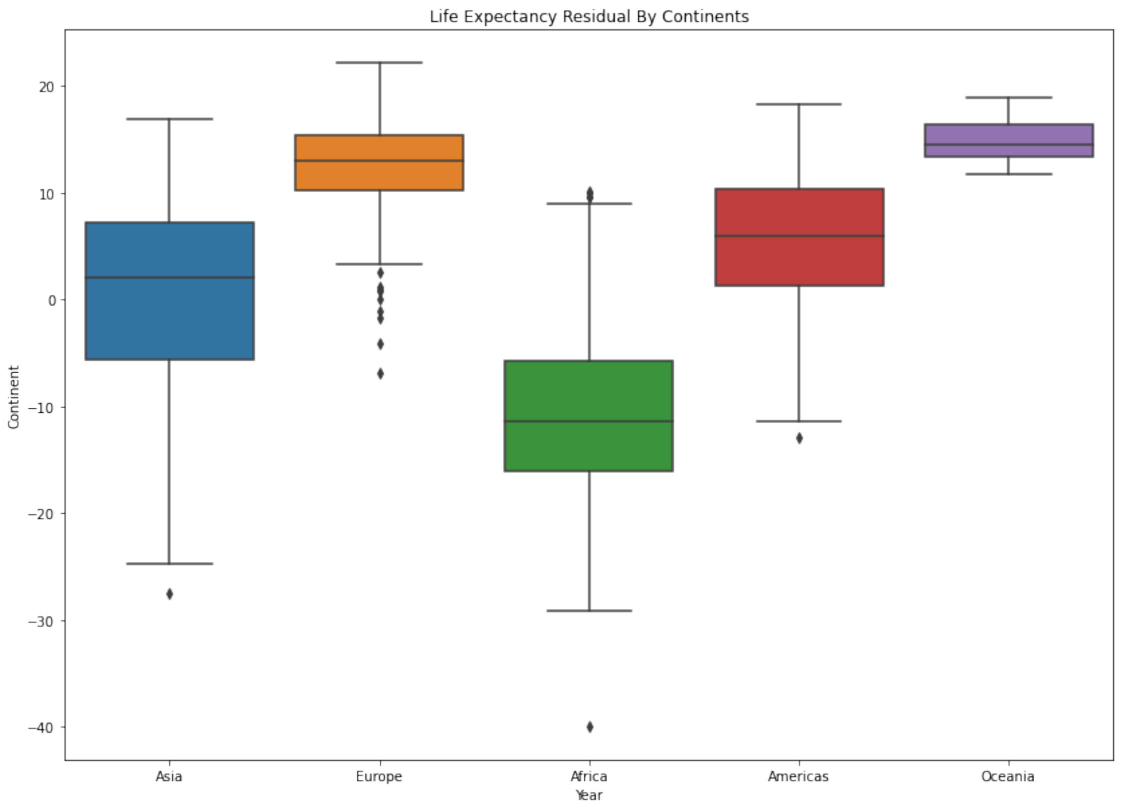
##Question 7 We reject the null hypothesis, because the p value is nearly 0. This means the there is nearly 0 possibility of the null hypothesis being true.

[ ]:
```
## Exercise 3

# Using the model fitted above to predict life expectancy for each year.

predictions = {}
for year in years_list:
  predictions[year] = mdl.predict([[year]])[0]

# Iterate through the data for each year and subtracting observed life␣
 ↪expectancy by the predicted one to get the residual for each data entry.

data['residual'] = data.apply(lambda row: row['lifeExp'] -␣
 ↪predictions[row['year']], axis=1)

# Group residual data by years and appending them into array for violin plot

violin_list2 = []

for year in years_list:
  filtered = data.loc[data['year'] == year]['residual']
  violin_list2.append(filtered)

plt.figure(figsize=(14,10))
plt.title('Life Expectancy Residual Over Years')
plt.xlabel('Year')
plt.ylabel('Residual')
plt.violinplot(violin_list2, years_list, widths=3, showmeans=True)
plt.show()
```



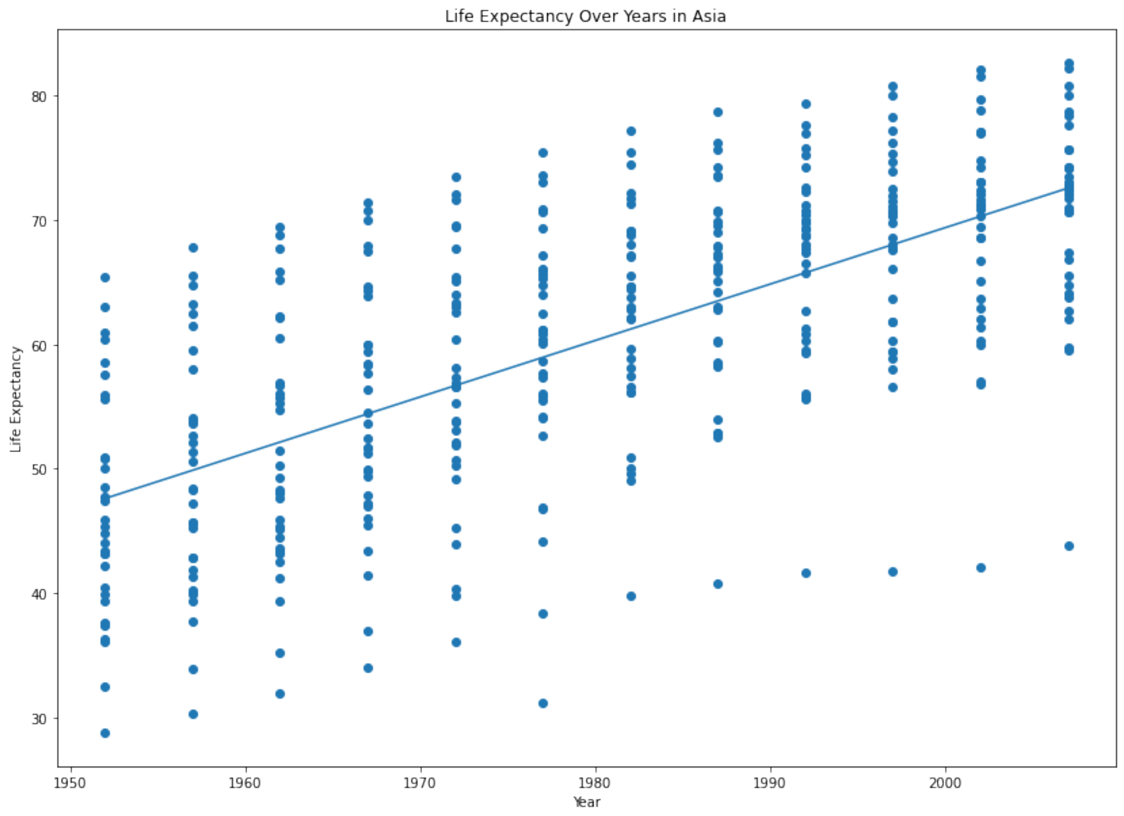
##Question 8 Yes, it is linear as I had expected, but it turns out that the violin plot is not noramlly distributed. It is similar to the original violin plot in that for most of the years, the data is skewed, not symmetric and multimodal.

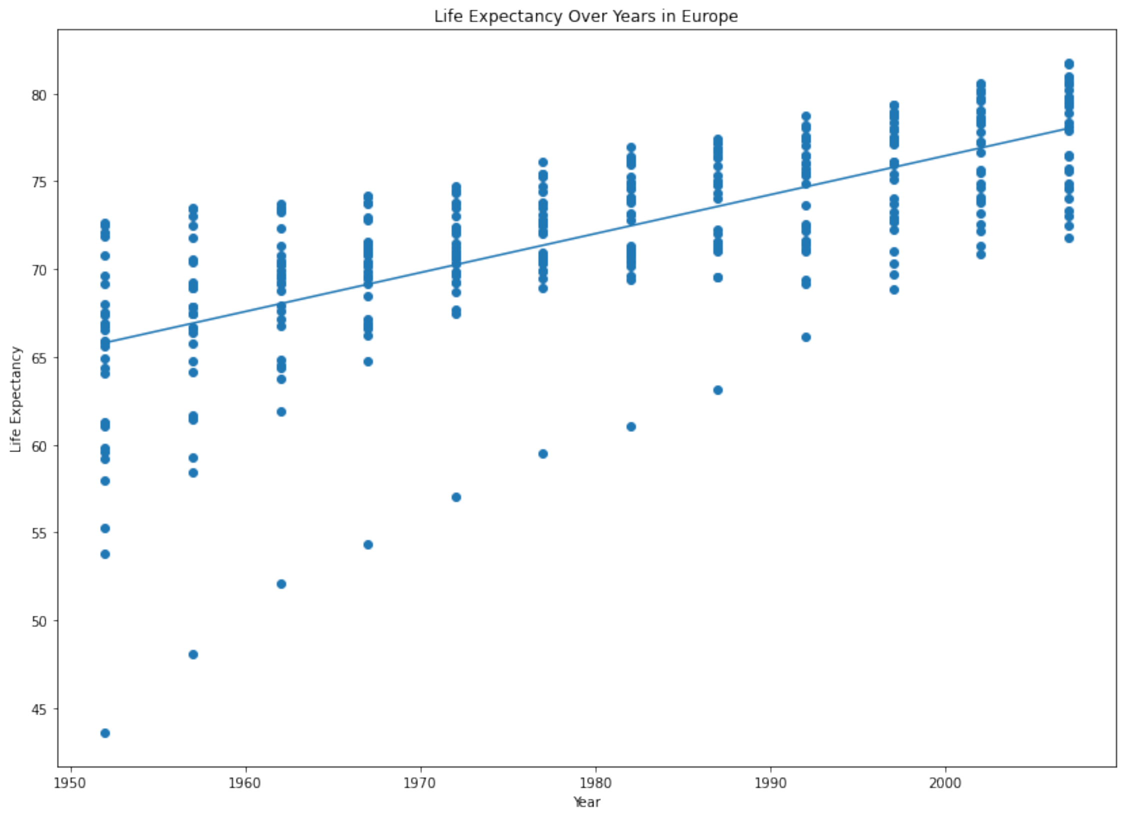
```
[ ]: ## Exercise 4

     # Making box plot with the same data used to generate the violin plot above␣
      ↪using Seaborn

     import seaborn as sbn

     plt.figure(figsize=(14,10))
     sbn.boxplot(x='continent', y='residual', data=data)
     plt.title('Life Expectancy Residual By Continents')
     plt.xlabel('Year')
     plt.ylabel('Continent')
     plt.show()
```

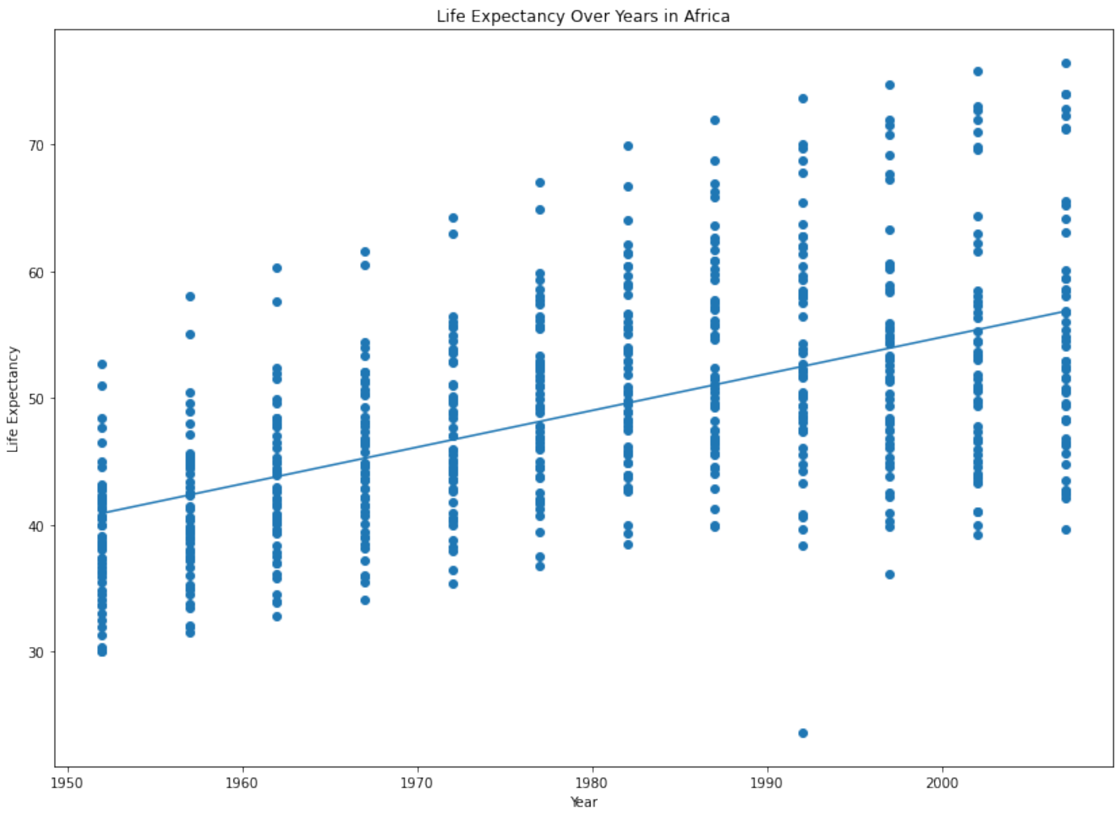


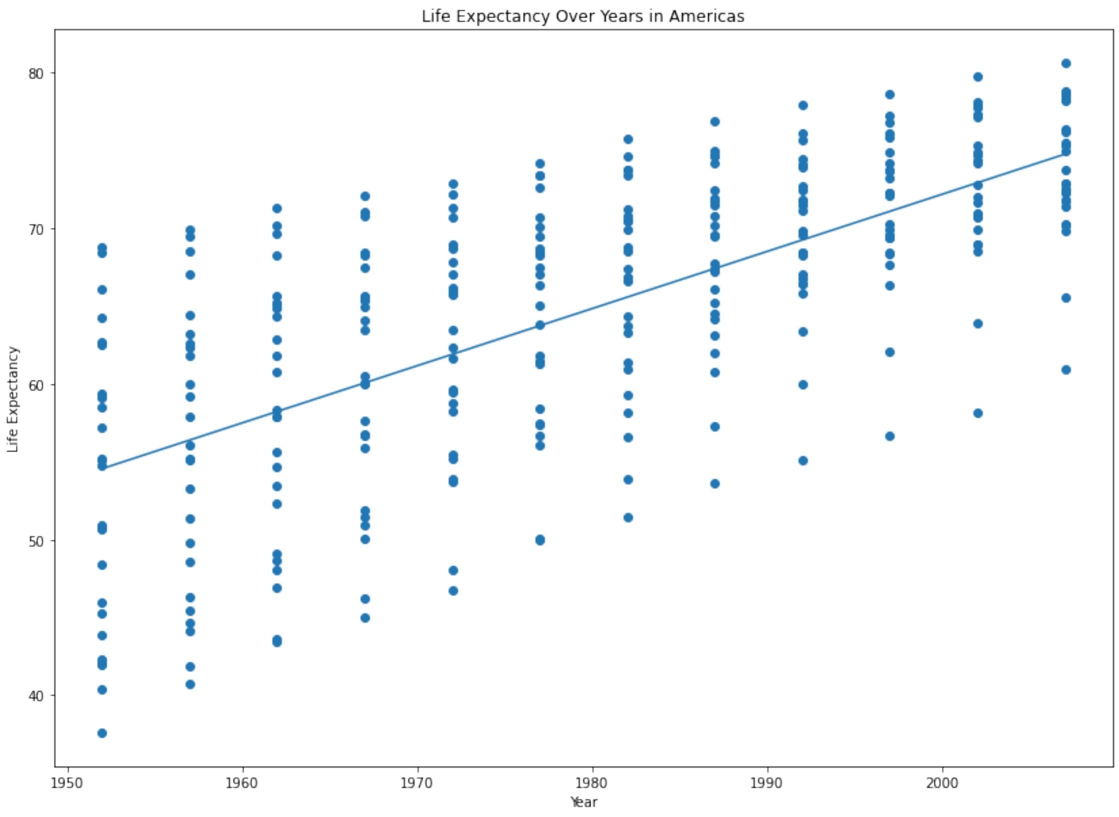
##Question 9 There is a dependence between residual and continent as each box plot varies greatly from continent to continent meaning that continents also influence life expectancy over time. This suggests that when performing a regression analysis of life expectancy across time, we need to also take the continents into consideration as another variable.

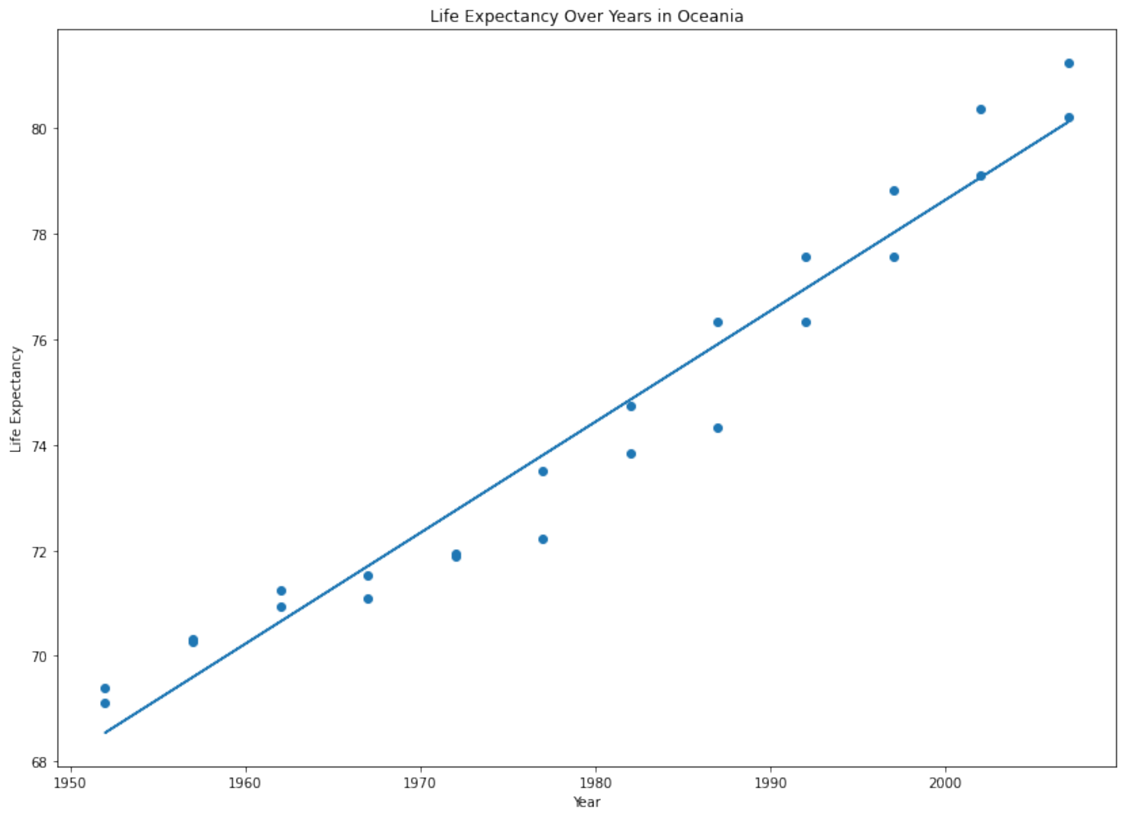
```
[ ]: ## Exercise 5

# Iterate through each continent filtering for only the data corresponding to␣
 ↪that continent
# For each continent, plot a scatter plot and line of best fit with the␣
 ↪filtered data

continents_list = data['continent'].unique()

for continent in continents_list:

# filtering for data on current continent
  filtered = data.loc[data['continent'] == continent]
  x = filtered['year']
  y = filtered['lifeExp']

  plt.figure(figsize=(14,10))
  plt.title('Life Expectancy Over Years in {}'.format(continent))
  plt.xlabel('Year')
  plt.ylabel('Life Expectancy')
  plt.scatter(x, y)

  a, b = np.polyfit(x, y, 1)
  plt.plot(x, a * x + b)

  plt.show()
```




Life Expectancy Over Years in Europe
Life Expectancy
Year
80
75
70
65
60
55
50
45
1950
1960
1970
1980
1990
2000


Life Expectancy Over Years in Africa
Life Expectancy
Year
30
40
50
60
70
1950
1960
1970
1980
1990
2000


Life Expectancy Over Years in Americas
Life Expectancy
Year
80
70
60
50
40
1950
1960
1970
1980
1990
2000



##Question 10 Yes, because the regression line and data vary greatly for each continent. From the 5 scatter plots above, one for each continent, it is clear that the year played no role in this variance. This means that the continent also plays a role in the overall life expectancy over time.

[ ]:
```
## Exercise 6

# Get summary of linear regression model between life expectancy and year with␣
 ↪continent as an interaction term

mdl2 = sm.ols(formula = 'lifeExp ~ year * continent', data=data).fit()
print(mdl2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                lifeExp   R-squared:                       0.693
Model:                            OLS   Adj. R-squared:                  0.691
Method:                 Least Squares   F-statistic:                     424.3
Date:                Thu, 01 Dec 2022   Prob (F-statistic):               0.00
Time:                        01:02:02   Log-Likelihood:                -5771.9
No. Observations:                1704   AIC:                         1.156e+04
Df Residuals:                    1694   BIC:                         1.162e+04
Df Model:                           9
```

```
Covariance Type:            nonrobust
================================================================================
==============
                                   coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------------
--------------
Intercept                     -524.2578     32.963    -15.904      0.000
-588.911    -459.605
continent[T.Americas]         -138.8484     57.851     -2.400      0.016
-252.315     -25.382
continent[T.Asia]             -312.6330     52.904     -5.909      0.000
-416.396    -208.870
continent[T.Europe]            156.8469     54.498      2.878      0.004
49.957     263.737
continent[T.Oceania]           182.3499    171.283      1.065      0.287
-153.599     518.298
year                             0.2895      0.017     17.387      0.000
0.257       0.322
year:continent[T.Americas]       0.0781      0.029      2.673      0.008
0.021       0.135
year:continent[T.Asia]           0.1636      0.027      6.121      0.000
0.111       0.216
year:continent[T.Europe]        -0.0676      0.028     -2.455      0.014
-0.122      -0.014
year:continent[T.Oceania]       -0.0793      0.087     -0.916      0.360
-0.249       0.090
==============================================================================
Omnibus:                       27.121   Durbin-Watson:                   0.242
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               44.106
Skew:                          -0.121   Prob(JB):                     2.65e-10
Kurtosis:                       3.750   Cond. No.                     2.09e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 2.09e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

##Question 11 Most of the parameters in the model are significantly different from zero as they have p-values less that 0.05 indicating that the observed relationship is statistically significant and therefore the null hypothesis should be rejected. However, Oceania with and without interaction have p-values greater than 0.05 being 0.289 and 0.360 respectively implying their parameters could possibly be zero.

```
[ ]: mdl2.params
[ ]: Intercept                        -524.257846
     continent[T.Americas]            -138.848447
     continent[T.Asia]                -312.633049
     continent[T.Europe]               156.846852
     continent[T.Oceania]              182.349883
     year                                0.289529
     year:continent[T.Americas]          0.078122
     year:continent[T.Asia]              0.163593
     year:continent[T.Europe]           -0.067597
     year:continent[T.Oceania]          -0.079257
     dtype: float64
```

##Question 12 Due to **statsmodels.formula.api.ols** dropping Africa's one hot encoding column, it becomes a reference variable and parameters for other continents are relative to Africa now.

Average change in life expectancy per year by continent:

Africa: 0.289529 years

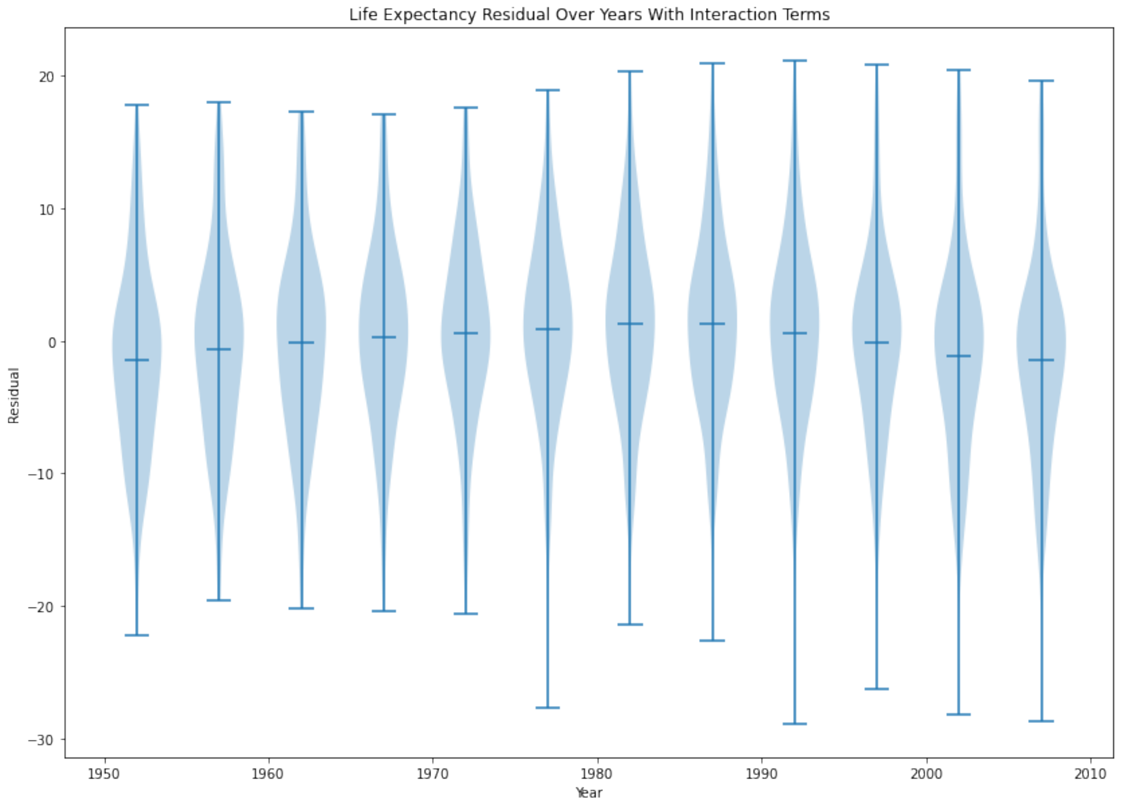
Americas: 0.078122 + 0.289529 = 0.367651 years

Asia: 0.163593 + 0.289529 = 0.453122 years

Europe: -0.067597 + 0.289529 = 0.221932 years

Oceania: -0.079257 + 0.289529 = 0.210272 years

```
[ ]: ## Exercise 7

     # Using the new model with interation term, generate a new set of predictions
     # Subtract observed life expectancy by the new prediction to get the new␣
      ↪residual

     predictions2 = mdl2.predict()

     data['residual2'] = data['lifeExp'] - predictions2

     violin_list3 = []

     # Group residual data by years and appending them into array for violin plot

     for year in years_list:
       filtered = data.loc[data['year'] == year]['residual2']
       violin_list3.append(filtered)

     plt.figure(figsize=(14,10))
     plt.title('Life Expectancy Residual Over Years With Interaction Terms')
     plt.xlabel('Year')
     plt.ylabel('Residual')
     plt.violinplot(violin_list3, years_list, widths=3, showmeans=True)
plt.show()
```



The violin plots are now normally distributed so it fits with the original assumptions of the linear regression model.

#Part 2: Classification For this part of the project, I used the Wine recognition dataset provided by SKLearn. This dataset contains 178 samples consisting of 13 different measurements taken from a chemical analysis of wines produced in the same region in Italy by three cultivators. More info on it here: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html#sklearn.datasets.load_wine

The independent variabels for this data set are which will all be used as predictors: - Alcohol - Malic acid - Ash - Alcalinity of ash - Magnesium - Total phenols - Flavanoids - Nonflavanoid phenols - Proanthocyanins - Color intensity - Hue - OD280/OD315 of diluted wines - Proline

The outcome we are predicting is which class each wine belongs to which SKLearn divides into: - class_0 - class_1 - class_2

For this project, I used K-Nearest-Neightbor and Decision Trees as my classifiers for the data. I used GridSearch with 10-fold cross-validation to tune the hyperparameters for my classifiers by maximizing the accuracy metric.

For KNN, I used GridSearch to tune the most accurate k value from a range of 1 to 20. I also included Minkowski metric to determine whether to use Manhattan Distance (p = 1) or Euclidean

Distance (p = 2). The hyperparameters that resulted in the highest accuracy after 10-fold cross-validation were k = 11 and p = 1.

For Decision Tree, I simply used GridSearch to find the max depth that will maximize my accuracy from a range of 1 to 10. The max depth that resulted in the highest accuracy after 10-fold cross-validation was 5.

[ ]:
```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV
from sklearn.preprocessing import StandardScaler
import math

# Loading data from SKLearn's dataset

ml_data = load_wine()
df = pd.DataFrame(ml_data.data, columns = ml_data.feature_names)

# Adding the targets to the dataframe
df['target'] = ml_data.target
df
```

[ ]:
```
     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0      14.23        1.71  2.43               15.6      127.0           2.80
1      13.20        1.78  2.14               11.2      100.0           2.65
2      13.16        2.36  2.67               18.6      101.0           2.80
3      14.37        1.95  2.50               16.8      113.0           3.85
4      13.24        2.59  2.87               21.0      118.0           2.80
..       …         …   …                …        …             …
173    13.71        5.65  2.45               20.5       95.0           1.68
174    13.40        3.91  2.48               23.0      102.0           1.80
175    13.27        4.28  2.26               20.0      120.0           1.59
176    13.17        2.59  2.37               20.0      120.0           1.65
177    14.13        4.10  2.74               24.5       96.0           2.05

     flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0          3.06                  0.28             2.29             5.64  1.04
1          2.76                  0.26             1.28             4.38  1.05
2          3.24                  0.30             2.81             5.68  1.03
3          3.49                  0.24             2.18             7.80  0.86
4          2.69                  0.39             1.82             4.32  1.04
..          …                   …                …              …   …
173        0.61                  0.52             1.06             7.70  0.64
174        0.75                  0.43             1.41             7.30  0.70
175        0.69                  0.43             1.35            10.20  0.59
176        0.68                  0.53             1.46             9.30  0.60
177        0.76                  0.56             1.35             9.20  0.61
```

```
     od280/od315_of_diluted_wines  proline  target
0                            3.92   1065.0       0
1                            3.40   1050.0       0
2                            3.17   1185.0       0
3                            3.45   1480.0       0
4                            2.93    735.0       0
..                            …        …      …
173                          1.74    740.0       2
174                          1.56    750.0       2
175                          1.56    835.0       2
176                          1.62    840.0       2
177                          1.60    560.0       2

[178 rows x 14 columns]
```

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier

# Remove targets from dataframe to separate training and test data.
target = df.pop('target')
x = df
y = target

# Shuffle data and split 20% of data for testing later
x_train, x_test, y_train, y_test = train_test_split(x, y, stratify = y,␣
 ↪test_size = 0.2, random_state = 30)

# Tuning number of neighbors and Minkowski metric
# Using GridSearchCV to exhaustively test hyperparameters with 10-fold␣
 ↪cross-validation for maximum accuracy
knn = KNeighborsClassifier()
n_neighbors = list(range(1,21))
hyper = dict(n_neighbors = n_neighbors, p = [1, 2])

clf = GridSearchCV(knn, param_grid = hyper, scoring = 'accuracy', cv = 10)
best_mdl = clf.fit(x_train, y_train)
print('Neighbors:', best_mdl.best_estimator_.get_params()['n_neighbors'])
print('P: ', best_mdl.best_estimator_.get_params()['p'])
print('Best Accuracy: ', best_mdl.best_score_)

# Creating new KNN classifier using optimal hyperparamters found above
# Score the classifier using the test data that was split ealier (holdout) and␣
 ↪10-fold cross validation
best_knn = KNeighborsClassifier(n_neighbors = 11, p = 1)
best_knn.fit(x_train, y_train)
knn_score = best_knn.score(x_test, y_test)
knn_cross_scores = cross_val_score(best_knn, x, y, scoring = 'accuracy', cv =␣
 ↪10)
```

```python
print('Test Set Score: ', knn_score)
print('Average 10-Fold Cross Validation Score: ', knn_cross_scores.mean())

# Calculating standard error
std_error = knn_cross_scores.std() / (math.sqrt(len(ml_data)))
print('Standard Error: ', std_error)
```

```
Neighbors: 11
P:  1
Best Accuracy:  0.7971428571428572
Test Set Score:  0.7222222222222222
Average 10-Fold Cross Validation Score:  0.7643790849673202
Standard Error:  0.023951149609l9839
```

[ ]:
```python
from sklearn import tree

# Shuffle data and split 20% of data for testing later
x_train, x_test, y_train, y_test = train_test_split(x, y, stratify = y,␣
 ↪test_size = 0.2, random_state = 30)

# Tuning max depth hyperparameter
# Using GridSearchCV to exhaustively test hyperparameters with 10-fold␣
 ↪cross-validation for maximum accuracy
dtree = tree.DecisionTreeClassifier(random_state = 0)
max_depth = list(range(1, 11))
hyper2 = dict(max_depth = max_depth)

clf2 = GridSearchCV(dtree, param_grid = hyper2, scoring = 'accuracy', cv = 10)
best_mdl2 = clf2.fit(x_train, y_train)
print('Max Depth: ', best_mdl2.best_estimator_.get_params()['max_depth'])
print('Best Accuracy: ', best_mdl2.best_score_)

# Creating a new Decision Tree classifier using optimal max depth parameter␣
 ↪found earlier
# Score the classifier using the test data that was split ealier (holdout) and␣
 ↪10-fold cross validation
best_dtree = tree.DecisionTreeClassifier(random_state = 0, max_depth = 5)
best_dtree.fit(x_train, y_train)
dtree_score = best_dtree.score(x_test, y_test)
dtree_cross_scores = cross_val_score(best_dtree, x, y, scoring = 'accuracy', cv␣
 ↪= 10)
print('Test Set Score: ', dtree_score)
print('Average 10-Fold Cross Validation Score: ', dtree_cross_scores.mean())

# Calculating standard error
std_error2 = dtree_cross_scores.std() / (math.sqrt(len(ml_data)))
print('Standard Error: ', std_error2)
```

```
Max Depth:  5
Best Accuracy:  0.9166666666666666
Test Set Score:  0.8611111111111112
Average 10-Fold Cross Validation Score:  0.8705882352941178
Standard Error:  0.03270804873974641
```

As seen, using accuracy as my metric, it appears that the Decision Tree model generated from the data scored higher the K-Nearest-Neighbor model in both holdout validation and average 10-fold cross-validation by around 10-14%. However, the Decision Tree has a slightly higher standard error when compared to the KNN model. Overall, it appears that the Decision Tree model is a better classifier for this dataset.