Discussion:

https://www.kaggle.com/kumarajarshi/life-expectancy-who/discussion

https://www.kaggle.com/suryanarayanjena/life-expectancy-india

Run Jupyter Notebook

1. Run: jupyter notebook in the Run

Life Expectancy India

- Explain about Corelation
- Correlation and Regression

Simple Regression

Step 1: *importing required libraries*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

- Numpy: NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
- Pandas: pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python.
- Matplotplot & pyplot: matplotlib. pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc

- Seaborn: Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- Warnings:

Step 2: read file and check data correlation heatmap

```
#read file and check data correlation heatmap
data = pd.read_csv('lifeexpectancydata.csv')
sns.heatmap(data.corr())
plt.show()
```

Step 3: Identifying linearly paired data

```
#Trying to identify linearly paired data
sns.scatterplot(data=data, x="GDP",y="Life
expectancy")
plt.show()
sns.scatterplot(data=data, x="Schooling",y="Life
expectancy")
plt.show()
sns.scatterplot(data=data, x="Income composition of
resources",y="Life expectancy")
plt.show()
sns.scatterplot(data=data, x="Income composition of
resources",y="Schooling")
plt.show()

data.loc[data['Country'] == "India"]
#India Data Correlation Heatmap
indata = data.loc[data['Country'] == "India"]
sns.heatmap(indata.corr())
plt.show()
sns.scatterplot(data=indata, x="GDP",y="Life
expectancy")
```

```
plt.show()
sns.scatterplot(data=indata, x="Year",y="Life
expectancy")
plt.show()
```

Step 4: *Modeling Year vs Life expectancy*

```
###Modeling Year vs Life expectancy
X = indata.iloc[:, 1:2].values  #Year
#y = dataset.iloc[:, 3].values
Y=indata['Life expectancy'].values
```

iloc returns a Pandas Series when one row is selected, and a Pandas DataFrame when multiple rows are selected, or if any column in full is selected.

Step 5: *Splitting the dataset into the Training set and Test set*

```
# Splitting the dataset into the Training set and
Test set
from sklearn.model_selection import
train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, Y, test_size = 0.25,
random_state = 0)
```

The train-**test split** procedure is used to estimate the performance of machine **learning** algorithms when they are used to make predictions on **data** not used to train the model.

Step 6:

```
#Visualization
sns.scatterplot(data=indata, x="GDP", y="Life
expectancy")
plt.show()
```

Step 7: Fitting the data

```
# Fitting Simple Linear Regression to the Training
set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Step 8: Running the Model

```
#intercept value: c
print("Intercept value: ",regressor.intercept_)
#coef as m
print("Coefficient value: ",regressor.coef_)
#predicting for 2021 year
pred_2021 = regressor.coef_ * 2021 +
regressor.intercept_
print(" Life Expectancy for the year 2021 is: ",
pred_2021)
```

Step 9: Predicting

```
# Predicting the Test set results
```

```
y_pred = regressor.predict(X_test)
print("Prediction for X Test Data: ",y_pred)
```

Step 10: Calculating Accuracy Metric

For regression algorithms, three evaluation metrics are commonly used:

1. Mean Absolute Error (MAE) measures how close the predictions are to the actual outcomes; thus, a lower score is better. It is the mean of the absolute value of the errors. It is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

2. Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

3. Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors. creates a single value that summarizes the error in the model. By squaring the difference, the metric disregards the difference between over-prediction and under-prediction. Formula:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

1. Relative absolute error (RAE) is the relative absolute difference between expected and actual values; relative because the mean difference is divided by the arithmetic mean.
2. Relative squared error (RSE) similarly normalizes the total squared error of the predicted values by dividing by the total squared error of the actual values.
3. Mean Zero One Error (MZOE) indicates whether the prediction was correct or not. In other words: ZeroOneLoss(x,y) = 1 when x!=y; otherwise 0.
4. Coecient of determination, often referred to as R2, represents the predictive power of the model as a value between 0 and 1. Zero means the model is random (explains nothing); 1 means there is a perfect t. However, caution should be used in interpreting R2 values, as low values can be entirely normal and high values can be suspect.

The Scikit-Learn library comes with pre-built functions that can be used to find out these values for us. Let's find the values for these metrics using our test data:

```
import numpy as np
from sklearn import metrics
explained_variance=metrics.explained_variance_score
```

```
(y_test, y_pred)
mean_absolute_error=metrics.mean_absolute_error(y_t
est, y_pred)
mse=metrics.mean_squared_error(y_test, y_pred)
mean_squared_log_error=metrics.mean_squared_log_err
or(y_test, y_pred)
median_absolute_error=metrics.median_absolute_error
(y_test, y_pred)
r2=metrics.r2_score(y_test, y_pred)
print('Explained_variance: ',
round(explained_variance,2))
print('Mean_Squared_Log_Error: ',
round(mean_squared_log_error,2))
print('R-squared: ', round(r2,4))
print('Mean Absolute Error(MAE): ',
round(mean_absolute_error,2))
print('Mean Squared Error (MSE): ', round(mse,2))
print('Root Mean Squared Error (RMSE): ',
round(np.sqrt(mse),2))
```

Multiple Linear Regression

Step 1: *importing required libraries*

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

- Numpy:
- Pandas
- Matplotplot & pyplot

- Seaborn
- warnings

Step 2: *read file and check data correlation heatmap*

```
#read file and check data correlation heatmap
data = pd.read_csv('lifeexpectancydata.csv')
sns.heatmap(data.corr())
plt.show()
```

Step 3: Identifying linearly paired data

```
data.loc[data['Country'] == "India"]
#India Data Correlation Heatmap
indata = data.loc[data['Country'] == "India"]
sns.heatmap(indata.corr())
plt.show()
sns.scatterplot(data=indata, x="Year",y="Life
expectancy")
plt.show()
sns.scatterplot(data=indata, x="Adult
Mortality",y="Life expectancy")
plt.show()
sns.scatterplot(data=indata, x="infant
deaths",y="Life expectancy")
plt.show()
```

Step 4: *Modeling Year vs Life expectancy*

```
###Modeling Year vs Life expectancy
X = indata.iloc[:,[1,4,5]].values  #Year
#y = dataset.iloc[:, 3].values
Y=indata['Life expectancy'].values
```

Step 5: *Splitting the dataset into the Training set and Test set*

```
# Splitting the dataset into the Training set and
Test set
from sklearn.model_selection import
train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, Y, test_size = 0.25,
random_state = 0)
```

Step 6:

```
#Visualization
sns.scatterplot(data=indata, x="Year", y="Life
expectancy")
plt.show()
sns.scatterplot(data=indata, x="Adult Mortality",
y="Life expectancy")
plt.show()
sns.scatterplot(data=indata, x="infant deaths",
y="Life expectancy")
plt.show()
```

Step 7: Fitting the data

```
# Fitting Multiple Linear Regression to the
Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Step 8: Running the Model

```
#intercept value: c
print("Intercept value: ",regressor.intercept_)
#coef as m
print("Coefficient value: ",regressor.coef_)
#predicting for 2021 year
pred_2021 = regressor.coef_[0] * 2021 +
regressor.coef_[1] * 291 + regressor.coef_[2] * 91+
regressor.intercept_
print(" Life Expectancy for the year 2021 is: ",
pred_2021)
```

Step 9: Predicting

```
# Predicting the Test set results
y_pred = regressor.predict(X_test)
print("Prediction for X Test Data: ",y_pred)
```

Step 10: Calculating Accuracy Metric

```
import numpy as np
```

```
from sklearn import metrics
explained_variance=metrics.explained_variance_score
(y_test, y_pred)
mean_absolute_error=metrics.mean_absolute_error(y_t
est, y_pred)
mse=metrics.mean_squared_error(y_test, y_pred)
mean_squared_log_error=metrics.mean_squared_log_err
or(y_test, y_pred)
median_absolute_error=metrics.median_absolute_error
(y_test, y_pred)
r2=metrics.r2_score(y_test, y_pred)
print('Explained_variance: ',
round(explained_variance,2))
print('Mean_Squared_Log_Error: ',
round(mean_squared_log_error,2))
print('R-squared: ', round(r2,4))
print('Mean Absolute Error(MAE): ',
round(mean_absolute_error,2))
print('Mean Squared Error (MSE): ', round(mse,2))
print('Root Mean Squared Error (RMSE): ',
round(np.sqrt(mse),2))
```

Step 11: OLS Summary

```
from statsmodels.api import OLS
import statsmodels.api as sm
#In our model, y will be dependent on 2 values:
coefficienct
# and constant, so we need to add additional column
in X for
#constant value
X = sm.add_constant(X)
summ = OLS(Y, X).fit().summary()
print("Summary of the dataset: \n",summ)
```

Interpretation

- The regression line with equation [Y = 3.0859+ (9.9893*X1)], is helpful to predict the value of the Y variable from the given value of the X1 variable.
- The slope (9.9893) represents the change in the Y per unit change in the X1 variable. It means that the value Y increases by 9.9893 with each unit increase in X1
- The intercept (3. 0859) represents the value of Y at X1 = 0. Here need to be cautious to interpret intercept as sometimes the value (X1=0) does not make any sense (e.g. speed of car or height of the person). In such cases, the values within the range of X1 should be considered to interpret the intercept
- The coefficient of determination (r-squared) is 0.874 (87.4%), which suggests that 87.4% of the variance in Y can be explained by X1 alone. Adjusted r-squared is useful where there are multiple X variables in the model (multiple linear regression)
- The correlation coefficient (r) is 0.935 (square root of r-squared), which suggests that there is a positive relationship between X and Y variables. As X1 increases, Y also increases
- From Regression Coefficients, the P-value obtained for X1 independent variable is significant (<0.05), it suggests that X1 significantly influence the response variable Y (there is a significant relationship between X1 and Y)
- From ANOVA, the P-value is significant (<0.05), which suggests that there is a significant relationship between X1 and Y. The X1 variable can reliably predict the Y variable.

Exaplanation:

This summary provides quite a lot of information about the fit. The parts of the table we think are the most important are **bolded** in the description below.

The left part of the first table provides basic information about the model fit:

Element	Description
Dep. Variable	Which variable is the response in the model (dependent variable)
Model	What model you are using in the fit

Element	Description
Method	How the parameters of the model were calculated
No. Observations	The number of observations
DF Residuals	Degrees of freedom of the residuals. Number of observations – number of parameters
DF Model	Number of parameters in the model (not including the constant term if present)

The right part of the first table shows the goodness of fit

Element	Description
R-squared	**The coefficient of determination. A statistical measure of how well the regression line approximates the real data points.**
Adj. R-squared	**The above value adjusted based on the number of observations and the degrees-of-freedom of the residuals**
F-statistic	A measure how significant the fit is. The mean squared error of the model divided by the mean squared error of the residuals
Prob (F-statistic)	The probability that you would get the above statistic, given the null hypothesis that they are unrelated
Log-likelihood	The log of the likelihood function.
AIC	The Akaike Information Criterion. Adjusts the log-likelihood based on the number of observations and the complexity of the model.
BIC	The Bayesian Information Criterion. Similar to the AIC, but has a higher penalty for models with more parameters.

The second table reports for each of the coefficients

Metric	Description
coef	The estimated value of the coefficient
std err	The basic standard error of the estimate of the coefficient. More sophisticated errors are also available.
t	t-statistic value. This is a measure of how statistically significant the coefficient is.
P > |t|	P-value that the null-hypothesis that the coefficient = 0 is true. If it is less than the confidence level, often 0.05, it indicates that there is a statistically significant relationship between the term and the response.
[95.0% Conf. Interval]	The lower and upper values of the 95% confidence interval

Finally, there are several statistical tests to assess the distribution of the residuals

Element	Description
Skewness	A measure of the symmetry of the data about the mean. Normally-distributed errors should be symmetrically distributed about the mean (equal amounts above and below the line).
Kurtosis	A measure of the shape of the distribution. Compares the amount of data close to the mean with those far away from the mean (in the tails).
Omnibus	D'Angostino's test. It provides a combined statistical test for the presence of skewness and kurtosis.
Prob(Omnibus)	The above statistic turned into a probability
Jarque-Bera	A different test of the skewness and kurtosis

Element	Description
Prob (JB)	The above statistic turned into a probability
Durbin-Watson	A test for the presence of autocorrelation (that the errors are not independent.) Often important in time-series analysis
Cond. No	A test for multicollinearity (if in a fit with multiple parameters, the parameters are related with each other).

Detailed explanation:

Residuals: Residuals are essentially the difference between the actual observed response values (Marks) and the response values that the model predicted. When assessing how well the model fit the data, you should look for a symmetrical distribution across these points on the mean value zero (0).

Coefficients: Theoretically, in simple linear regression, the coefficients are two unknown constants that represent the intercept and slope terms in the linear model. If we wanted to predict the Marks obtained given the hours of study, we would get a training set and produce estimates of the coefficients to then use it in the model formula.

Coefficient - Estimate: Without studying, one can score on average 20.7583 marks - that's Intercept. The second row in the Coefficients is the slope, saying that for every 1-hour increase in the study, the marks goes up by 7.5675.

Coefficient - Standard Error: The coefficient Standard Error measures the average amount that the coefficient estimates vary from the actual average value of our response variable. We'd ideally want a lower number relative to its coefficients. In our example, we've previously determined that 1-hour increase in the study, the marks goes up by 7.5675. The Standard Error can be used to compute an estimate of the expected difference in case we ran the model again and again. In other words, we can say that the Marks obtained can vary by 0.3009. The Standard Errors can also be used to compute confidence intervals and to statistically test the hypothesis of the existence of a relationship between Hours of study and marks obtained.

Coefficient - t value: The coefficient t-value is a measure of how many standard deviations our coefficient estimate is far away from 0. We want it to be far away from zero as this would indicate we could reject the null hypothesis - that is, we could declare a relationship between Hours and Marks exist. In our example, the t-statistic values are relatively far away from zero and are large relative to the standard error, which could indicate a relationship exists. In general, t-values are also used to compute p-values.

Coefficient - Pr(>t): The Pr(>t) acronym found in the model output relates to the probability of observing any value equal or larger than t. A small p-value indicates that it is unlikely we will observe a relationship between the predictor (Hours) and response (Marks) variables due to chance. Typically, a p-value of 5% or less is a good cut-off point. In our model example, the p-values are very close to zero. Note the 'signif. Codes' associated to each estimate. Three stars (or asterisks) represent a highly significant p-value. Consequently, a small p-value for the intercept and the slope indicates that we can reject the null hypothesis which allows us to conclude that there is a relationship between speed and distance.

Residual Standard Error: Residual Standard Error is measure of the quality of a linear regression fit. Theoretically, every linear model is assumed to contain an error term E. Due to the presence of this error term, we are not capable of perfectly predicting our response variable (Marks) from the predictor (Hours) one. The Residual Standard Error is the average amount that the response (Marks) will deviate from the true regression line. In our example, the actual distance required to stop can deviate from the true regression line by approximately 4.599, on average. Simplistically, degrees of freedom are the number of data points that went into the estimation of the parameters used after taking into account these parameters (restriction). *Degree of Freedom* is given by the difference between the number of observations in the sample and the number of variables in the model.

Multiple R-squared, Adjusted R-squared: The R-squared (R2R2) statistic provides a measure of how well the model is fitting the actual data. It takes the form of a proportion of variance. R2R2 is a measure of the linear relationship between our predictor variable (Hours) and our response / target variable (Marks). It always lies between 0 and 1 (i.e.: a number near 0 represents a regression that does not explain the variance in the response variable well and a number close to 1 does explain the observed variance in the response variable). In our example, the R2R2 we get is 0.9576. Or roughly 95% of the variance found in the response variable (Marks) can be explained by the predictor variable (Hours). It's hard to define what level of R2R2 is appropriate to claim the model fits well. Essentially, it will vary with the application and the domain studied.

A side note: In multiple regression settings, the R2R2 will always increase as more variables are included in the model. That's why the adjusted R2R2 is the preferred measure as it adjusts for the number of variables considered.

F-Statistic: F-statistic is a good indicator of whether there is a relationship between our predictor and the response variables. The further the F-statistic is from 1 the better it is. However, how much larger the F-statistic needs to be depends on both the number of data points and the number of predictors. Generally, when the number of data points is large, an F-statistic that is only a little bit larger than 1 is already sufficient to reject the null hypothesis (H0 : There is no relationship between Hours and Marks). The reverse is true as if the number of data points is small, a large F-statistic is required to be able to ascertain that there may be a relationship between predictor and response variables. In our example the F-statistic is 632.4 which is relatively larger than 1 given the size of our data.

COMPLETE PROGRAMS

```
#importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
#read file and check data correlation heatmap
data = pd.read_csv('lifeexpectancydata.csv')
sns.heatmap(data.corr())
plt.show()
#Trying to identify linearly paired data
sns.scatterplot(data=data, x="GDP",y="Life expectancy")
plt.show()
sns.scatterplot(data=data, x="Schooling",y="Life expectancy")
plt.show()
sns.scatterplot(data=data, x="Income composition of resources",y="Life
expectancy")
plt.show()
sns.scatterplot(data=data, x="Income composition of
resources",y="Schooling")
plt.show()

data.loc[data['Country'] == "India"]
#India Data Correlation Heatmap
indata = data.loc[data['Country'] == "India"]
sns.heatmap(indata.corr())
plt.show()
sns.scatterplot(data=indata, x="GDP",y="Life expectancy")
plt.show()
```

```python
sns.scatterplot(data=indata, x="Year",y="Life expectancy")
plt.show()
###Modeling Year vs Life expectancy
X = indata.iloc[:, 1:2].values  #Year
#y = dataset.iloc[:, 3].values
Y=indata['Life expectancy'].values

##Run Regression algo
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.25, random_state = 0)

#Visualization
sns.scatterplot(data=indata, x="GDP", y="Life expectancy")
plt.show()
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

#intercept value: c
print("Intercept value: ",regressor.intercept_)
#coef as m
print("Coefficient value: ",regressor.coef_)
#predicting for 2021 year
pred_2021 = regressor.coef_ * 2021 + regressor.intercept_
print(" Life Expectancy for the year 2021 is: ", pred_2021)

# Predicting the Test set results
y_pred = regressor.predict(X_test)
print("Prediction for X Test Data: ",y_pred)
import numpy as np
from sklearn import metrics
explained_variance=metrics.explained_variance_score(y_test, y_pred)
mean_absolute_error=metrics.mean_absolute_error(y_test, y_pred)
mse=metrics.mean_squared_error(y_test, y_pred)
mean_squared_log_error=metrics.mean_squared_log_error(y_test, y_pred)
median_absolute_error=metrics.median_absolute_error(y_test, y_pred)
r2=metrics.r2_score(y_test, y_pred)
print('Explained_variance: ', round(explained_variance,2))
print('Mean_Squared_Log_Error: ', round(mean_squared_log_error,2))
print('R-squared: ', round(r2,4))
print('Mean Absolute Error(MAE): ', round(mean_absolute_error,2))
print('Mean Squared Error (MSE): ', round(mse,2))
print('Root Mean Squared Error (RMSE): ', round(np.sqrt(mse),2))

#########  Multiple Regression

###Modeling Year vs Life expectancy
X = indata.iloc[:, [1,4,5]].values  #Year
```

```
#y = dataset.iloc[:, 3].values
Y=indata['Life expectancy'].values

##Run Regression algo
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.25, random_state = 0)

#Visualization
sns.scatterplot(data=indata, x="Year", y="Life expectancy")
plt.show()
sns.scatterplot(data=indata, x="Adult Mortality", y="Life expectancy")
plt.show()
sns.scatterplot(data=indata, x="infant deaths", y="Life expectancy")
plt.show()
# Fitting Simple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)

#intercept value: c
print("Intercept value: ",regressor.intercept_)
#coef as m
print("Coefficient value: ",regressor.coef_)
#predicting for 2021 year
pred_2021 = regressor.coef_[0] * 2021 + regressor.coef_[1] * 291 +
regressor.coef_[2] * 91+ regressor.intercept_
print(" Life Expectancy for the year 2021 is: ", pred_2021)

# Predicting the Test set results
y_pred = regressor.predict(X_test)
print("Prediction for X Test Data: ",y_pred)
import numpy as np
from sklearn import metrics
explained_variance=metrics.explained_variance_score(y_test, y_pred)
mean_absolute_error=metrics.mean_absolute_error(y_test, y_pred)
mse=metrics.mean_squared_error(y_test, y_pred)
mean_squared_log_error=metrics.mean_squared_log_error(y_test, y_pred)
median_absolute_error=metrics.median_absolute_error(y_test, y_pred)
r2=metrics.r2_score(y_test, y_pred)
print('Explained_variance: ', round(explained_variance,2))
print('Mean_Squared_Log_Error: ', round(mean_squared_log_error,2))
print('R-squared: ', round(r2,4))
print('Mean Absolute Error(MAE): ', round(mean_absolute_error,2))
print('Mean Squared Error (MSE): ', round(mse,2))
print('Root Mean Squared Error (RMSE): ', round(np.sqrt(mse),2))

from statsmodels.api import OLS
import statsmodels.api as sm
#In our model, y will be dependent on 2 values: coefficienct
```

```
# and constant, so we need to add additional column in X for
#constant value
X = sm.add_constant(X)
summ = OLS(Y, X).fit().summary()
print("Summary of the dataset: \n",summ)
```