WEEK-04: Polynomial and Multiple Regression

Machine Learning Model evaluation metrics

The various ways to check the performance of our machine learning or deep learning model and why to use one in place of the other. We will discuss terms like:

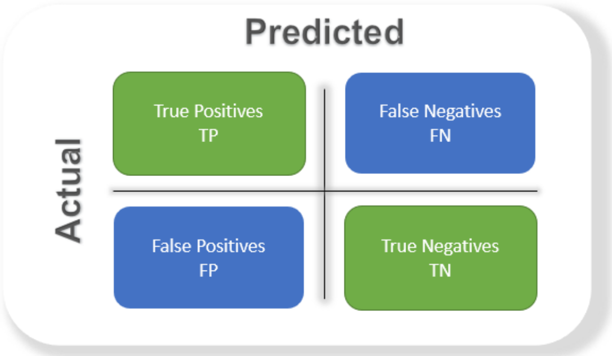
- Confusion matrix
- Accuracy
- Precision
- Recall
- Specificity
- F1 score
- Precision-Recall or PR curve
- **ROC** (**R**eceiver **O**perating **C**haracteristics) curve
- PR vs ROC curve.

For simplicity, we will mostly discuss things in terms of a binary classification problem where let's say we'll have to find if an image is of a cat or a dog. Or a patient is having cancer (positive) or is found healthy (negative). Some common terms to be clear with are:

True positives (TP): Predicted positive and are actually positive.
False positives (FP): Predicted positive and are actually negative.
True negatives (TN): Predicted negative and are actually negative.
False negatives (FN): Predicted negative and are actually positive.

Confusion matrix

It's just a representation of the above parameters in a matrix format. Better visualization is always good :)



Accuracy

The most commonly used metric to judge a model and is actually not a clear indicator of the performance. The worse happens when classes are imbalanced.

$$\frac{TP + TN}{TP + FP + TN + FN}$$

Take for example a cancer detection model. The chances of actually having cancer are very low. Let's say out of 100, 90 of the patients don't have cancer and the remaining 10 actually have it. We don't want to miss on a patient who is having cancer but goes undetected (false negative). Detecting everyone as not having cancer gives an accuracy of 90% straight. The model did nothing here but just gave cancer free for all the 100 predictions.

We surely need better alternatives.

Precision

Percentage of positive instances out of the ***total predicted positive*** instances. Here denominator is the model prediction done as positive from the whole given dataset. Take it as to find out '*how much the model is right when it says it is right*'.

$$\frac{TP}{TP + FP}$$

Recall/Sensitivity/True Positive Rate
Percentage of positive instances out of the ***total actual positive*** instances. Therefore denominator (*TP + FN)* here is the *actual* number of positive instances present in the dataset. Take it as to find out '*how much extra right ones, the model missed when it showed the right ones'.*

$$\frac{TP}{TP + FN}$$

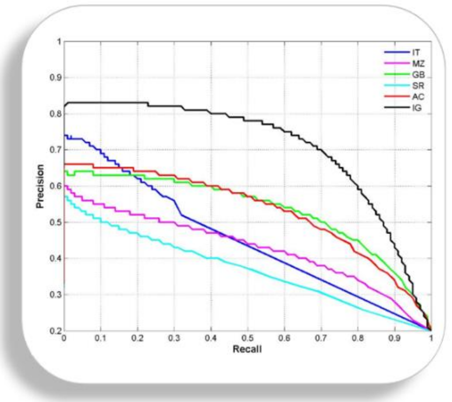
Specificity
Percentage of negative instances out of the ***total actual negative*** instances. Therefore denominator (*TN + FP)* here is the *actual* number of negative instances present in the dataset. It is similar to recall but the shift is on the negative instances. *Like finding out how many healthy patients were not having cancer and were told they don't have cancer*. Kind of a measure to see how separate the classes are.

$$\frac{TN}{TN + FP}$$

F1 score
It is the harmonic mean of precision and recall. This takes the contribution of both, so higher the F1 score, the better. See that due to the product in the numerator if one goes low, the final F1 score goes down significantly. So a model does well in F1 score if the positive predicted are actually positives (precision) and doesn't miss out on positives and predicts them negative (recall).

$$\frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 * precision * recall}{precision + recall}$$

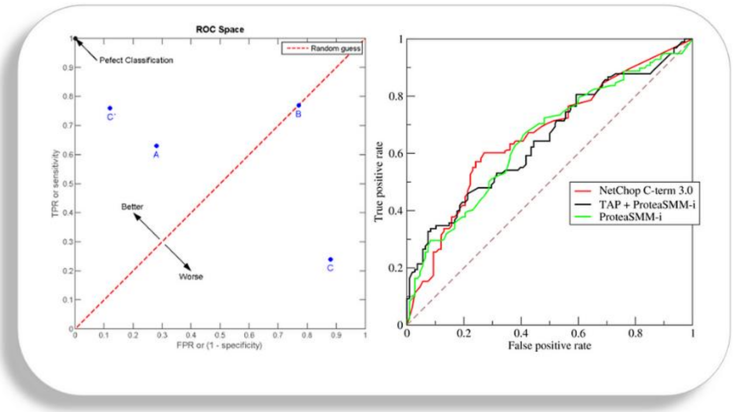
One drawback is that both precision and recall are given equal importance due to which according to our application we may need one higher than the other and F1 score may not be the exact metric for it. Therefore either weighted-F1 score or seeing the PR or ROC curve can help.
PR curve
It is the curve between precision and recall for various threshold values. In the figure below we have 6 predictors showing their respective precision-recall curve for various threshold values. The top right part of the graph is the ideal space where we get high precision and recall. Based on our application we can choose the predictor and the threshold value. PR AUC is just the area under the curve. The higher its numerical value the better.



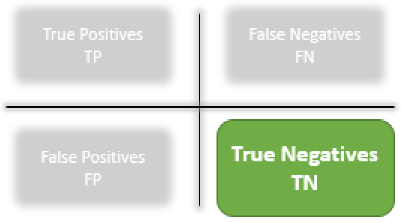
ROC curve
ROC stands for receiver operating characteristic and the graph is plotted against TPR and FPR for various threshold values. As TPR increases FPR also increases. As you can see in the first figure, we have four categories and we want the threshold value that leads us closer to the top left corner. Comparing different predictors (here 3) on a given dataset also becomes easy as you can see in figure 2, one can choose the threshold according to the application at hand. ROC AUC is just the area under the curve, the higher its numerical value the better.

$$\textit{True Positive Rate (TPR) = RECALL} = \frac{TP}{TP+FN}$$

$$\textit{False Positive Rate (FPR) = 1} - Specificity = \frac{FP}{TN+FP}$$



PR vs ROC curve
Both the metrics are widely used to judge a models performance.
Which one to use PR or ROC?



The answer lies in TRUE NEGATIVES.
Due to the absence of TN in the precision-recall equation, they are useful in imbalanced classes. In the case of class imbalance when there is a majority of the negative class. The metric doesn't take much into consideration the high number of TRUE NEGATIVES of the negative class which is in majority, giving better resistance to the imbalance. This is important when the detection of the positive class is very important.
Like to detect cancer patients, which has a high class imbalance because very few have it out of all the diagnosed. We certainly don't want to miss on a person having cancer and going undetected (recall) and be sure the detected one is having it (precision).
Due to the consideration of TN or the negative class in the ROC equation, it is useful when both the classes are important to us. Like the detection of cats and dog. The importance of true negatives makes sure that both the classes are given importance, like the output of a ML/DL model in determining the image is of a cat or a dog.

Classification
Here the goal is to learn a mapping from inputs x to outputs y, where y ∈ {1,...,C}, with C being the number of classes. If C = 2, this is called binary classification (in which case we often assume y ∈ {0, 1}); if C > 2, this is called multiclass classification. If the class labels are not mutually exclusive (e.g., somebody may be classified as tall and strong), we call it multi-label classification, but this is best viewed as predicting multiple related binary class labels (a so-called multiple output model). When we use the term "classification", we will mean multiclass classification with a single output, unless we state otherwise.
One way to formalize the problem is as function approximation. We assume y = f(x) for some unknown function f, and the goal of learning is to estimate the function f given a labeled training set, and then to make predictions using $\hat{y} = \hat{f}(\mathbf{x})$. (We use the hat symbol to denote an estimate.) Our main goal is to make predictions on novel inputs, meaning ones that we have not seen before (this is called generalization), since predicting the response on the training set is easy.

LINEAR REGRESSION

Regression analysis may broadly be defined as the analysis of relationships among variables. This relationship is given as an equation that helps to predict the dependent variable Y through
one or more independent variables. In regression analysis, the variable whose values vary with the variations in the values of the other variable(s) is called the **dependent variable** or **response variable**. The other variables which are independent in nature and influence the response variable are called **independent variables**, **predictor variables** or **regressor variables**.

Example: Suppose a statistician employed by a cold drink bottler is analysing the product delivery and service operation for vending machines. He would like to find how the delivery time taken by the delivery man to load and service a machine is related to the volume of delivery cases. The statistician visits 50 randomly chosen retailer shops having vending machines and observes the delivery time (in minutes) and the volume of delivery cases for each shop. He plots those 50 observations on a graph, which shows that an approximate linear relationship exists between the delivery time and delivery volume. If Y represents the delivery time and X, the delivery volume, the equation of a straight line relating these two variables may be given as

$$\mathbf{Y = a + bX \ldots (1)}$$

where a is the intercept and b, the slope.

In such cases, we draw a straight line in the form of equation (1) so that the data points generally fall near the straight line. Now, suppose the points do not fall exactly on the straight line. Then we should modify equation (1) to minimise the difference between the observed value of Y and that given by the straight line (a + b X). This is known as **error**.

The error e, which is the difference between the observed value and the predicted value of the variable of interest Y, may be conveniently assumed as a statistical error. This error term accounts for the variability in Y that cannot be explained by the linear relationship between X and Y. It may arise due to the effects of other factors. Thus, a more plausible model for the variable of interest (Y) may be given as

$$\mathbf{Y = a + b X + e \ldots (2)}$$

where the intercept a and the slope b are unknown constants and e is a random error component.
Equation (2) is called a **linear regression model**.

Fitting of regression line

Let the given data of n pairs of observations on X and Y be as follows:

$$X: X_1\ X_2\ X_3 \ldots \ldots X_i \ldots \ldots X_n$$
$$Y: Y_1\ Y_2\ Y_3 \ldots \ldots Y_i \ldots \ldots Y_n$$

where Y is the dependent variable and X, the independent variable.
Suppose, we wish to fit the following simple regression equation to the data: Y = a + bX
where a is the intercept and b is the slope of the equation.

For fitting equation to the data on (X, Y), we follow the steps given below:

Step 1: We draw a scatter diagram by plotting the (X, Y) points given in data.

Step 2: We construct a table as given below and take the sum of the values of X_i, Y_i, X_iY_i, and X_i^2.
We write the values of $\sum X$, $\sum Y$, $\sum XY$ and $\sum X^2$ in the last row.

Step 3: We express of $\hat{a}$ given in equation (1) as follows:

$$\hat{a} = \bar{Y} - b\bar{X} = \frac{1}{n}\left[\sum Y - b\sum X\right]$$

$$\hat{b} = \frac{n\sum XY - \sum X \sum Y}{n\sum X^2 - \left(\sum X\right)^2}$$

Where

substitute above values in the regression equation and get

$$\hat{Y} = \hat{a} + \hat{b}X$$

POLYNOMIAL REGRESSION

Some engineering data is poorly represented by a straight line. For these cases, a curve would be better suited to it these data. The method is to it polynomials to the data using polynomial regression.

Polynomial Regression is a form of regression analysis in which the relationship between the independent variables and dependent variables are modeled in the nth degree polynomial. Polynomial Regression models are usually fit with the method of least squares. The **least square method** minimizes the variance of the coefficients, under the Gauss Markov Theorem. Polynomial Regression is a special case of Linear Regression where we fit the polynomial equation on the data with a curvilinear relationship between the dependent and independent variables. A Quadratic Equation is a Polynomial Equation of 2nd Degree. However, this degree can increase to nth values.

The least-squares procedure can be readily extended to it the data to a higher-order polynomial. For example, suppose that we it a second-order polynomial or quadratic:

$$y = a_0 + a_1 x + a_2 x^2 + e$$

Where x-independent variable, y-dependent variable, a0,a1 and a2 are coefficients, and e –error term.

For this case the sum of the squares of the residuals is

$$S_r = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i - a_2 x_i^2)^2$$

we take the derivative with respect to each of the unknown coefficients of the polynomial, as in

$$\frac{\partial S_r}{\partial a_0} = -2 \sum (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum x_i (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

$$\frac{\partial S_r}{\partial a_2} = -2 \sum x_i^2 (y_i - a_0 - a_1 x_i - a_2 x_i^2)$$

These equations can be set equal to zero and rearranged to develop the following set of normal equations:

$$(n)a_0 + \left(\sum x_i\right)a_1 + \left(\sum x_i^2\right)a_2 = \sum y_i$$

$$\left(\sum x_i\right)a_0 + \left(\sum x_i^2\right)a_1 + \left(\sum x_i^3\right)a_2 = \sum x_i y_i$$

$$\left(\sum x_i^2\right)a_0 + \left(\sum x_i^3\right)a_1 + \left(\sum x_i^4\right)a_2 = \sum x_i^2 y_i$$

where all summations are from i = 1 through n. Note that the above three equations are linear and have three unknowns: a0 , a1 , and a2 . The coefficients of the unknowns can be calculated directly from the observed data. For this case, we see that the problem of determining a least-squares second-order polynomial is equivalent to solving a system of three simultaneous linear equations.

These equations can be rewritten in matrix form as follows:

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i^2 y_i \end{bmatrix}$$

Where:

- *n* is the number of data points.
- *xi* and *yi* are the values of the independent and dependent variables for the *i*-th data point.

The most common way to solve a system of linear equations is by using matrix algebra and methods like Gaussian elimination or matrix inversion. Here are the general steps to solve a system of linear equations:

Step 1: Write Down the System of Equations

Write the system of linear equations in standard form, where each equation has the form:

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = b$$

Where:

- $a1,a2,\ldots,an$ are the coefficients of the variables $x1,x2,\ldots,xn$.
- b is the constant on the right-hand side of the equation.

You should have as many equations as there are variables.

Step 2: Create a Coefficient Matrix

Construct a coefficient matrix A by extracting the coefficients of the variables from the left side of each equation. This matrix will have dimensions $n\times n$ if you have n variables.

Step 3: Create a Right-Hand Side Vector

Create a right-hand side vector B by extracting the constants from the right side of each equation. This vector will have dimensions $n\times 1$.

Step 4: Solve the System

There are several methods to solve the system:

There are several methods to solve the system:

- **Matrix Inversion**: If A is invertible (non-singular), you can solve the system using the formula $X=A^{-1}B$, where X is the vector of solutions.
- **Gaussian Elimination**: Use row operations to transform the augmented matrix $[A|B]$ into row-echelon or reduced row-echelon form. Then, back-substitute to find the values of the variables.
- **Matrix Factorization**: Factorize A into LU or QR form and use the factorization to solve the system more efficiently.
- **Numerical Methods**: For large or ill-conditioned systems, numerical methods like the Gauss-Seidel method or the Conjugate Gradient method are used.

Here's a simple example using matrix inversion in Python:

```
import numpy as np
# Define the coefficient matrix A and the right-hand side vector B
A = np.array([[2, 1], [1, 3]])
B = np.array([4, 7])
# Solve for the variables X using matrix inversion
X = np.linalg.inv(A).dot(B)
print("Solution:")
print(X)
```

The above code solves the system $2x_1+x_2=4$ and $x_1+3x_2=7$ and prints the values of x_1 and x_2 as the solution.

```
Solution:

[1. 2.]
```

Polynomial regression is a type of regression analysis in which the relationship between the independent variable (X) and the dependent variable (Y) is modeled as an nth-degree polynomial. In Python, you can perform polynomial regression using libraries like NumPy and scikit-learn. Here's a basic example of polynomial regression using scikit-learn:

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt

# Generate sample data
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
Y = 4 + 3 * X + np.random.randn(100, 1)

# Fit a polynomial regression model
degree = 2  # You can change the degree as needed
poly_features = PolynomialFeatures(degree=degree)
X_poly = poly_features.fit_transform(X)
```

```
model = LinearRegression()
model.fit(X_poly, Y)

# Make predictions
X_new = np.linspace(0, 2, 100).reshape(-1, 1)
X_new_poly = poly_features.transform(X_new)
Y_new = model.predict(X_new_poly)

# Plot the original data and the polynomial regression curve
plt.scatter(X, Y, label='Original Data')
plt.plot(X_new, Y_new, 'r-', label='Polynomial Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
# The coefficients of the multivariate polynomial regression model
coefficients = model.coef_
intercept = model.intercept_
print("Coefficients:")
print(coefficients)
print("Intercept:")
print(intercept)
```

In this example:

1. We generate some sample data points with random noise.
2. We use **PolynomialFeatures** from scikit-learn to transform our input features **X** into polynomial features up to a specified degree.
3. We then fit a linear regression model to the polynomial features.
4. Finally, we make predictions using the model and plot the original data along with the polynomial regression curve.

You can adjust the **degree** variable to control the degree of the polynomial you want to fit to your data. Higher-degree polynomials can capture more complex relationships but may also lead to overfitting, so be cautious when choosing the degree.

MULTIPLE LINEAR REGRESSION

Multiple linear regression is a method we can use to quantify the relationship between two or more predictor variables and a response variable.

The Regression Line: With one independent variable, we may write the regression equation as:

$$Y = a + bX + e$$

Where Y is an observed score on the dependent variable, *a* is the intercept, *b* is the slope, X is the observed score on the independent variable, and *e* is an error or residual.

We can extend this to any number of independent variables:

$$Y = a + b_1X_1 + b_2X_2 + \ldots + b_kX_k + e \quad (3.1)$$

Note that we have *k* independent variables and a slope for each. We still have one error and one intercept. Again we want to choose the estimates of *a* and *b* so as to minimize the sum of squared errors of prediction. The prediction equation is:

$$Y' = a + b_1X_1 + b_2X_2 + \ldots + b_kX_k \quad (3.2)$$

Finding the values of *b* (the slopes) is tricky for k>2 independent variables, and you really need matrix algebra to see the computations. It's simpler for k=2 IVs, which we will discuss here. But the basic ideas are the same no matter how many independent variables you have. If you understand the meaning of the slopes with two independent variables, you will likely be good no matter how many you have.

For the one variable case, the calculation of *b* and *a* was:

$$b = \frac{\sum xy}{\sum x^2}$$

$$a = \overline{Y} - b\overline{X}$$

For the two variable case:

$$b_1 = \frac{(\sum x_2^2)(\sum x_1 y) - (\sum x_1 x_2)(\sum x_2 y)}{(\sum x_1^2)(\sum x_2^2) - (\sum x_1 x_2)^2}$$

and

$$b_2 = \frac{(\sum x_1^2)(\sum x_2 y) - (\sum x_1 x_2)(\sum x_1 y)}{(\sum x_1^2)(\sum x_2^2) - (\sum x_1 x_2)^2}$$

where

- $\Sigma x_1^2 = \Sigma X_1^2 - ((\Sigma X_1)^2 / n)$
- $\Sigma x_2^2 = \Sigma X_2^2 - ((\Sigma X_2)^2 / n)$
- $\Sigma x_1 y = \Sigma X_1 y - ((\Sigma X_1 \Sigma y) / n)$
- $\Sigma x_2 y = \Sigma X_2 y - ((\Sigma X_2 \Sigma y) / n)$
- $\Sigma x_1 x_2 = \Sigma X_1 X_2 - ((\Sigma X_1 \Sigma X_2) / n)$

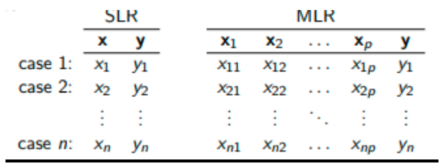
At this point, you should notice that all the terms from the one variable case appear in the two variable case. In the two variable case, the other X variable also appears in the equation. For example, X_2 appears in the equation for b_1. Note that terms corresponding to the variance of both X variables occur in the slopes. Also note that a term corresponding to the covariance of X1 and X2 (sum of deviation cross-products) also appears in the formula for the slope.
The equation for *a* with two independent variables is:

$$a = \overline{Y} - b_1\overline{X}_1 - b_2\overline{X}_2$$

This equation is a straight-forward generalization of the case for one independent variable.

MLR MATRIX FORMULATION

Multiple linear regression is a generalized form of simple linear regression, in which the data contains multiple explanatory variables.



	SLR		MLR				
	x	**y**	**x_1**	**x_2**	...	**x_p**	**y**
case 1:	x_1	y_1	x_{11}	x_{12}	...	x_{1p}	y_1
case 2:	x_2	y_2	x_{21}	x_{22}	...	x_{2p}	y_2
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
case n:	x_n	y_n	x_{n1}	x_{n2}	...	x_{np}	y_n

- For SLR, we observe pairs of variables.
- For MLR, we observe rows of variables.
- Each row (or pair) is called a case, a record, or a data point
- yi is the response (or dependent variable) of the ith observation
- There are p explanatory variables (or covariates, predictors, independent variables), and xik is the value of the explanatory variable xk of the ith case

$$y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i \quad \text{where } \varepsilon_i\text{'s are i.i.d. } N(0, \sigma^2)$$

In the model above,

- ε_i's (errors, or noise) are i.i.d. $N(0, \sigma^2)$
- Parameters include:

$$\begin{aligned} \beta_0 &= \text{intercept;} \\ \beta_k &= \text{regression coefficients (slope) for the } k\text{th} \\ &\quad \text{explanatory variable,} \quad k = 1, \ldots, p \\ \sigma^2 &= \text{Var}(\varepsilon_i) \text{ is the variance of errors} \end{aligned}$$

Refer further https://online.stat.psu.edu/stat462/node/132/

REGRESSION METRICS

Residual

The difference between the fitted value $\hat{Y}_i$ and Yi is known as the residual and is denoted by

$r_i = Y_i - \hat{Y}_i, \qquad i = 1, 2, \ldots, n$

The role of the residuals and its analysis is very important in regression modelling.

Mean Squared Error (MSE)

Mean squared error (MSE) measures the amount of error in statistical models. It assesses the average squared difference between the observed and predicted values. When a model has no error, the MSE equals zero. As model error increases, its value increases. The mean squared error is also known as the ***mean squared deviation (MSD)***.

For example, in regression, the mean squared error represents the average squared residual/error.

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

Where:
yi is the ith observed value.
ŷi is the corresponding predicted value.
n = the number of observations.

Squaring the error gives higher weight to the outliers, which results in a smooth gradient for small errors. Optimization algorithms benefit from this penalization for large errors as it is helpful in finding the optimum values for parameters. MSE will never be negative since the errors are squared. The value of the error ranges from zero to infinity. MSE increases exponentially with an increase in error. A good model will have an MSE value closer to zero.

Root Mean Square Error (RMSE)

Root Mean Squared Error (RMSE) is a popular metric used in machine learning and statistics to measure the accuracy of a predictive model. It quantifies the differences between predicted values and actual values, squaring the errors, taking the mean, and then finding the square root. RMSE provides a clear understanding of the model's performance, with lower values indicating better predictive accuracy.
RMSE is computed by taking the square root of MSE. RMSE is also called the Root Mean Square Deviation. It measures the average magnitude of the errors and is concerned with the deviations from the actual value. RMSE value with zero indicates that the model has a perfect fit. The lower the RMSE, the better the model and its predictions. A higher RMSE indicates that there is a large deviation from the residual to the ground truth. RMSE can be used with different features as it helps in figuring out if the feature is improving the model's prediction or not.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Ref: https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

Questions:

1. The data set of size n = 15 (Yield data) contains measurements of yield from an experiment done at five different temperature levels. The variables are y = yield and x = temperature in degrees Fahrenheit. The table below gives the data used for this analysis.

i	Temp.	Yield
1	50	3.3
2	50	2.8
3	50	2.9
4	70	2.3
5	70	2.6
6	70	2.1
7	80	2.5
8	80	2.9
9	80	2.4
10	90	3.0
11	90	3.1
12	90	2.8
13	100	3.3
14	100	3.5
15	100	3.0

a. Create a CSV file with sample data.
b. Write a Python function program to:
Find the fitted simple linear and polynomial regression equations for the given data.
c. Compare the coefficients obtained from manually intuitive and matrix formulation methods with your program.
d. Plot the scatterplot of the raw data and then another scatterplot with lines pertaining to a linear fit and a quadratic fit overlayed.
e. Compute the error, MSE, and RMSE.
Note: Do not use scikit-learn.

2. When heart muscle is deprived of oxygen, the tissue dies and leads to a heart attack ("myocardial infarction"). Apparently, cooling the heart reduces the size of the heart attack. It is not known, however, whether cooling is only effective if it takes place before the blood, flow to the heart becomes restricted. Some researchers (Hale, et al, 1997) hypothesized that cooling the heart would be effective in reducing the size of the heart attack even if it takes place after the blood flow becomes restricted.
To investigate their hypothesis, the researchers conducted an experiment on 32 anesthetized rabbits that were subjected to a heart attack. The researchers established three experimental groups:

- *Rabbits whose hearts were cooled to 6º C within 5 minutes of the blocked artery ("**early cooling**")*
- *Rabbits whose hearts were cooled to 6º C within 25 minutes of the blocked artery ("**late cooling**")*
- *Rabbits whose hearts were not cooled at all ("**no cooling**")*

*At the end of the experiment, the researchers measured the size of the **infarcted** (i.e., damaged) **area** (in grams) in each of the 32 rabbits. But, as you can imagine, there is great variability in the size of hearts. The size of a rabbit's infarcted area may be large only because it has a larger heart. Therefore, in order to adjust for differences in heart sizes, the researchers also measured the size of the **region at risk** for infarction (in grams) in each of the 32 rabbits.*

Infarc	***Area***	***Group***	***X2***	***X3***
0.119	*0.34*	*3*	*0*	*0*
0.19	*0.64*	*3*	*0*	*0*
0.395	*0.76*	*3*	*0*	*0*
0.469	*0.83*	*3*	*0*	*0*
0.13	*0.73*	*3*	*0*	*0*
0.311	*0.82*	*3*	*0*	*0*
0.418	*0.95*	*3*	*0*	*0*

0.48	*1.06*	*3*	*0*	*0*
0.687	*1.2*	*3*	*0*	*0*
0.847	*1.47*	*3*	*0*	*0*
0.062	*0.44*	*1*	*1*	*0*
0.122	*0.77*	*1*	*1*	*0*
0.033	*0.9*	*1*	*1*	*0*
0.102	*1.07*	*1*	*1*	*0*
0.206	*1.01*	*1*	*1*	*0*
0.249	*1.03*	*1*	*1*	*0*
0.22	*1.16*	*1*	*1*	*0*
0.299	*1.21*	*1*	*1*	*0*
0.35	*1.2*	*1*	*1*	*0*
0.35	*1.22*	*1*	*1*	*0*
0.588	*0.99*	*1*	*1*	*0*
0.379	*0.77*	*2*	*0*	*1*
0.149	*1.05*	*2*	*0*	*1*
0.316	*1.06*	*2*	*0*	*1*
0.39	*1.02*	*2*	*0*	*1*
0.429	*0.99*	*2*	*0*	*1*
0.477	*0.97*	*2*	*0*	*1*
0.439	*1.12*	*2*	*0*	*1*
0.446	*1.23*	*2*	*0*	*1*
0.538	*1.19*	*2*	*0*	*1*
0.625	*1.22*	*2*	*0*	*1*
0.974	*1.4*	*2*	*0*	*1*

With their measurements in hand, the researchers' primary research question was: Does the mean size of the infarcted area differ among the three treatment groups — no cooling, early cooling, and late cooling — when controlling for the size of the region at risk for infarction?
A regression model that the researchers might use in answering their research question is:

$$y_i=(\beta_0+\beta_1x_{i1}+\beta_2x_{i2}+\beta_3x_{i3})+\epsilon_i$$

where:

- y_i *is the size of the infarcted area (in grams) of rabbit i*
- x_{i1} *is the size of the region at risk (in grams) of rabbit i*
- x_{i2} *= 1 if early cooling of rabbit i, 0 if not*
- x_{i3} *= 1 if late cooling of rabbit i, 0 if not*

and the independent error terms ε_i *follow a normal distribution with mean 0 and equal variance* σ^2.
Illustrates the need for being able to "translate" a research question into a statistical procedure. Often, the procedure involves four steps, namely:

- *formulating a multiple regression model*
- *determining how the model helps answer the research question*
- *checking the model*
- *and performing a hypothesis test (or calculating a confidence interval)*

a. Create a CSV file with sample data.
b. Write a Python function program to:
c. Find the fitted multiple linear regression equation for the given data.
d. Compare the coefficients obtained manually using intuitive and matrix formulation methods with your program.
e. Plot the data adorned with the estimated regression equation.
f. Compute the error, MSE, and RMSE.
Note: Do not use scikit-learn.

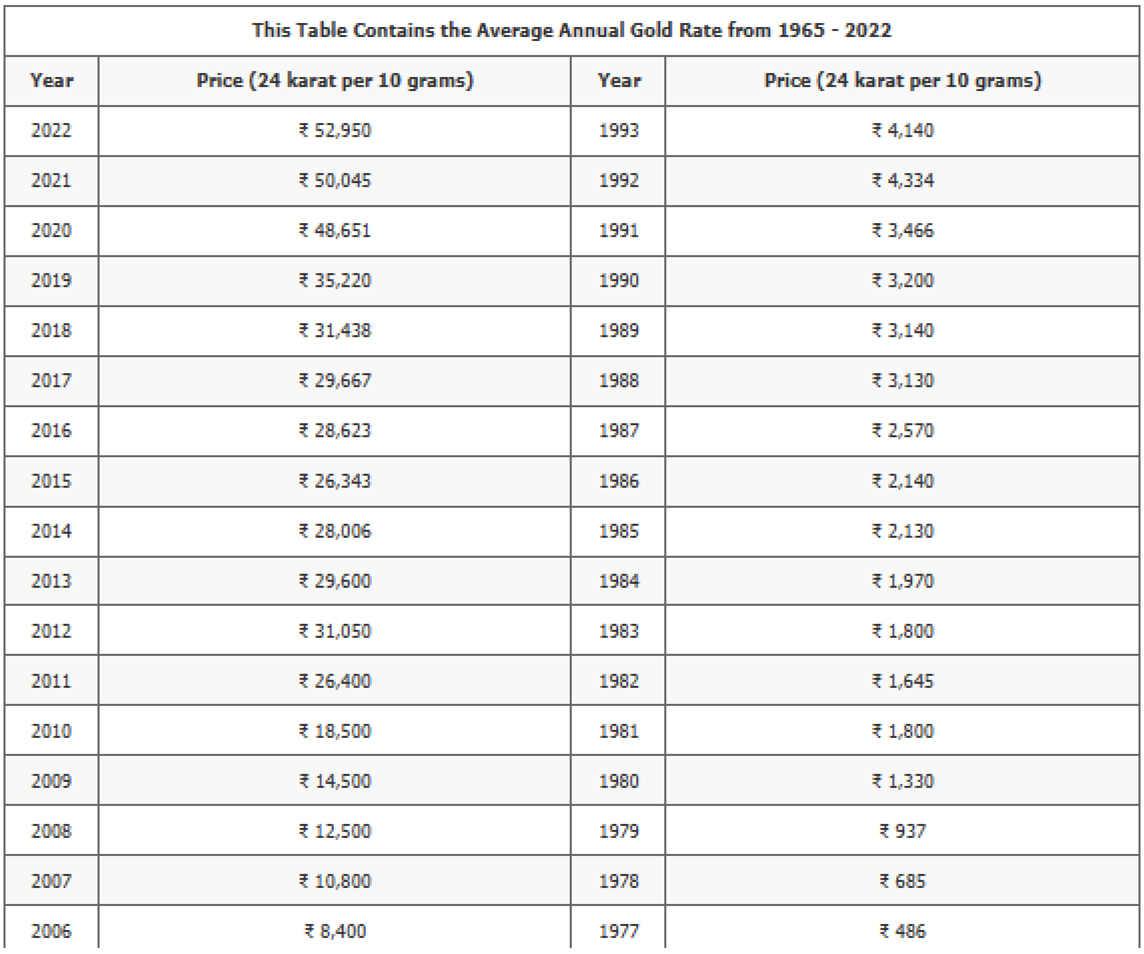
Additional questions

1. Consider the Table Contains the Average Annual Gold Rate from 1965 – 2022. Gold prices fluctuated throughout the year 2020 because of the COVID-19 epidemic. With gold functioning as a safe haven for investors, demand for the precious metal grew, and its price followed suit. During the epidemic, the stock market weakened, but it began to recover by the end of 2020 when the price of gold fell slightly.

It's crucial to remember that gold prices fluctuate during the year, and the figure below represents the average price for that year.

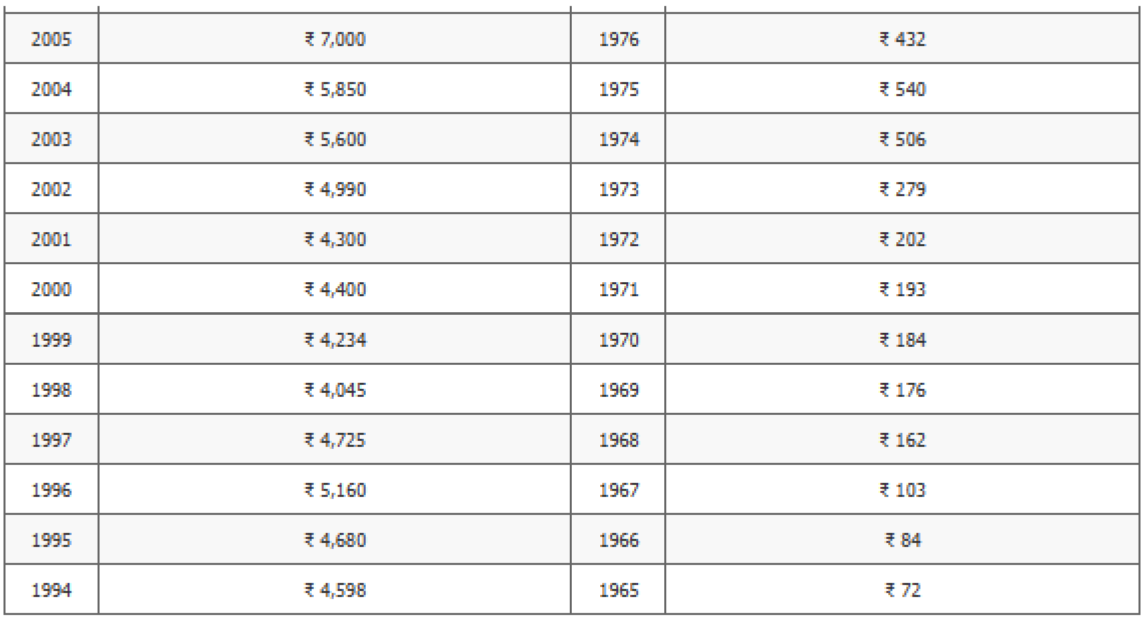
With the exception of a few lows shared across a few years, The table shows that the gold price trend has always been upward, supporting the claim that gold is a secure investment over extended periods of time.

Create CSV file and Write a python program to find the fitted simple linear regression equation for the given data. Compare the coefficients obtained from sklearn model with your program. Compute the error, MSE and RMSE. Predict the gold price with the year 2025 for 1 gram.



This Table Contains the Average Annual Gold Rate from 1965 - 2022

Year	Price (24 karat per 10 grams)	Year	Price (24 karat per 10 grams)
2022	₹ 52,950	1993	₹ 4,140
2021	₹ 50,045	1992	₹ 4,334
2020	₹ 48,651	1991	₹ 3,466
2019	₹ 35,220	1990	₹ 3,200
2018	₹ 31,438	1989	₹ 3,140
2017	₹ 29,667	1988	₹ 3,130
2016	₹ 28,623	1987	₹ 2,570
2015	₹ 26,343	1986	₹ 2,140
2014	₹ 28,006	1985	₹ 2,130
2013	₹ 29,600	1984	₹ 1,970
2012	₹ 31,050	1983	₹ 1,800
2011	₹ 26,400	1982	₹ 1,645
2010	₹ 18,500	1981	₹ 1,800
2009	₹ 14,500	1980	₹ 1,330
2008	₹ 12,500	1979	₹ 937
2007	₹ 10,800	1978	₹ 685
2006	₹ 8,400	1977	₹ 486



2005	₹ 7,000	1976	₹ 432
2004	₹ 5,850	1975	₹ 540
2003	₹ 5,600	1974	₹ 506
2002	₹ 4,990	1973	₹ 279
2001	₹ 4,300	1972	₹ 202
2000	₹ 4,400	1971	₹ 193
1999	₹ 4,234	1970	₹ 184
1998	₹ 4,045	1969	₹ 176
1997	₹ 4,725	1968	₹ 162
1996	₹ 5,160	1967	₹ 103
1995	₹ 4,680	1966	₹ 84
1994	₹ 4,598	1965	₹ 72

2. Consider the Question no 1 gold price with following year-wise silver price. Create a CSV file and Write a python program to find the fitted multiple linear regression equation for the given data. Compare the coefficients obtained from sklearn model with your program. Compute the error, MSE and RMSE. Predict the gold and silver price with the year 2024 for 1 gram.

Year	**Silver Rates in Rs./Kg.**	**Year**	**Silver Rates in Rs./Kg.**
1981	Rs.2715	2002	Rs.7875
1982	Rs.2720	2003	Rs.7695
1983	Rs.3105	2004	Rs.11770
1984	Rs.3570	2005	Rs.10675
1985	Rs.3955	2006	Rs.17405
1986	Rs.4015	2007	Rs.19520
1987	Rs.4794	2008	Rs.23625
1988	Rs.6066	2009	Rs.22165
1989	Rs.6755	2010	Rs.27255
1990	Rs.6463	2011	Rs.56900
1991	Rs.6646	2012	Rs.56290
1992	Rs.8040	2013	Rs.54030
1993	Rs.5489	2014	Rs.43070
1994	Rs.7124	2015	Rs.37825
1995	Rs.6335	2016	Rs.36990
1996	Rs.7346	2017	Rs.37825
1997	Rs.7345	2018	Rs.41400
1998	Rs.8560	2019	Rs.40600
1999	Rs.7615	2020	Rs.63435
2000	Rs.7900	2021	Rs.62572
2001	Rs.7215	2022	Rs.55100

3. Suppose you have a gold/silver price dataset with a single independent variable (X) and a dependent variable (Y). You want to fit a polynomial regression model to this data. Implement the process of selecting the appropriate degree for the polynomial (e.g., linear, quadratic, cubic) based on the dataset using Python.

4. Imagine you have a gold and silver price dataset with two independent variables (X1 and X2) and a dependent variable (Y). Implement in python, how you can perform multivariate polynomial regression to model the relationship between the independent variables and the dependent variable.