😊 Linear Regression and Evaluation Functions

In simple terms, **linear regression** is a machine learning algorithm used to make predictions about things that can have any value, like predicting someone's salary based on their years of experience.

How does it work?

Imagine you're trying to draw a straight line through a scatter plot of points (data). This line helps you predict the value of something based on one or more factors.

For example:

- If you want to predict a person's height based on their age, the line would help you estimate how tall someone might be at a certain age. The idea is that the closer the line is to the actual points (data), the more accurate your prediction will be.

In **simple linear regression**, you're working with just one factor (like age), and in **multiple linear regression**, you're working with more factors (like age, weight, and exercise habits).

Formula (in simple terms):

Linear regression uses an equation like: Prediction=Slope×Input+Intercept\text{Prediction} = \text{Slope} \times \text{Input} + \text{Intercept}Prediction=Slope×Input+Intercept

- **Slope**: How steep the line is (how much the prediction changes with the input)
- **Intercept**: Where the line crosses the Y-axis (the starting point of your prediction)

How do we check if it's a good prediction?

We have some methods to check how good our line is at making predictions:

1. **Mean Squared Error (MSE)**:
   Think of this as the average of all the errors we make in our predictions. Smaller numbers are better because that means our predictions are closer to the actual values.
2. **Mean Absolute Error (MAE)**:
   This is the average of how far off each prediction is from the actual answer, but instead of squaring the errors (like MSE), it just takes the difference. It's easier to understand because it's in the same units as the original data.
3. **R-squared**:
   This is like a score from 0 to 1 that tells you how well your line fits the data. A score of 1 means your line perfectly matches the data. A lower score means your line isn't doing a great job.

Why is it useful?

Linear regression is great when you want a simple, straightforward way to make predictions based on data. It helps you answer questions like:

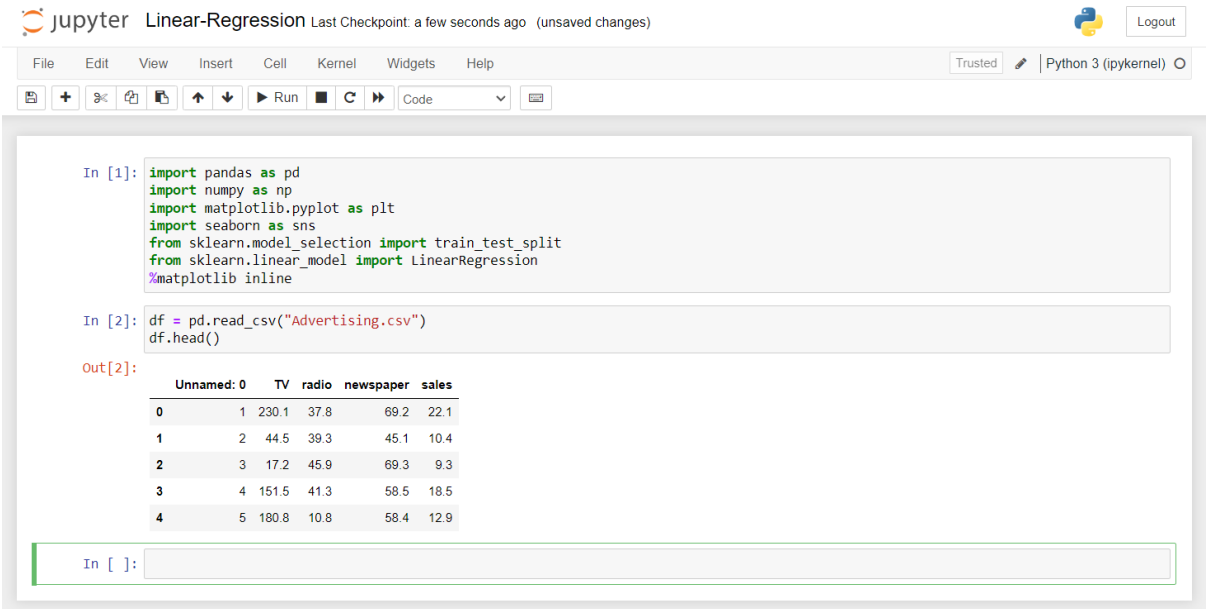
- "How much will sales increase if I spend more on advertising?"
- "What will the temperature be in the next hour based on current weather patterns?"

In summary, **linear regression** helps us draw a straight line through data so we can make educated guesses, and we have different ways to measure how good our guesses are!

😄 To begin with the Lab:

1. In your Jupyter Notebook create a new folder then you need to upload Advertising.csv file which you can find in the folder you get with the labs. Then you need to create a Python 3 Kernel in your Jupyter Notebook as you can see below.

2. Now we have imported some important libraries then we load our dataset as a data frame and then view some of its entries.
3. Also, you can see that in this data we have a column called Unnamed and we are going to drop this column because it is an invalid column in our dataset.



4. Below you can see that we dropped that column.

```
In [3]: df.drop(columns='Unnamed: 0',inplace=True)
        df.head()
```

Out[3]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

5. First, we need to identify the key parts of the dataset.
6. The input features are **TV, radio, and newspaper spending**, and the target is the **sales** column.
7. Our goal is to figure out how spending on TV, radio, and newspaper ads is related to the product's sales.
8. So, the **X values** (the inputs) are things like the spending on TV, radio, and newspaper ads, and the **Y value** (the output) is the sales.
   In statistics:
   - The inputs (X) are called **independent variables** because they are the things we control or measure.
   - The output (Y) is called the **dependent variable** because it depends on the inputs.
9. Another important point is that the target column, which is the output (sales), is made up of numbers. Since the output is numerical, the type of problem we are solving is called a **regression task**.
10. In our dataset, there are 200 rows in total. When we split the data into a 70/30 ratio, we get 140 rows for training (to learn from) and 60 rows for testing (to check how well the model works).

```
In [4]: df.shape
```

```
Out[4]: (200, 4)
```

```
In [5]: X = df.drop(columns='sales')
        y = df['sales']
        X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=4,test_size=0.3)
        X_train.shape,X_test.shape
```

```
Out[5]: ((140, 3), (60, 3))
```

11. Now, let's find the relationship between X (the inputs) and Y (the output). To do that, we use the command below.
12. After running this, the model will learn the relationship, and as a result, we'll get the coefficients, which show how each input affects the sales. We also get the intercept.

13. In this example, you can see that the machine learning algorithm took the data as input and generated parameters as the output. This means it has learned the patterns or relationships between **X** (the inputs) and **Y** (the output) from the dataset.

```
In [6]: lr = LinearRegression()
        lr.fit(X_train,y_train)

Out[6]: LinearRegression
        LinearRegression()
```

```
In [7]: lr.coef_

Out[7]: array([0.04533392, 0.17715767, 0.00553852])
```

```
In [8]: lr.intercept_

Out[8]: 3.090644364125108
```

14. Once we have trained the machine learning algorithm, we have our trained model. Using this model, we can generate outputs for new, unknown data.
15. When I refer to generating outputs on unknown data, I mean using a separate set of data, called test data, which we kept aside specifically for this purpose. This test data will help us see how well our model can predict results based on the patterns it learned during training.
16. Now, with the test data, I can make predictions. I can do this by using the command y_pred_test = lr.predict(X_test).
17. This will allow me to generate predictions based on the input values from **X_test**.

```
In [9]: X_test[:5]
```

Out[9]:

	TV	radio	newspaper
11	214.7	24.0	4.0
99	135.2	41.7	45.9
128	220.3	49.0	3.2
175	276.9	48.9	41.8
1	44.5	39.3	45.1

```
In [10]: y_pred_test = lr.predict(X_test)

In [11]: y_pred_test[:5]

Out[11]: array([17.09777441, 16.86148273, 21.7761552 , 24.53812602, 12.32008724])
```

18. When we evaluate the performance of the linear regression algorithm, we use evaluation functions to measure how well the model is doing. Depending on the specific task, we can choose the appropriate evaluation function.

For linear regression, some common evaluation metrics are:

Mean Absolute Error (MAE): This measures the average difference between the predicted values and the actual values, giving us an idea of how far off our predictions are.
Mean Squared Error (MSE): This measures the average of the squared differences between predicted and actual values, which helps to penalize larger errors more heavily.
Root Mean Squared Error (RMSE): This is simply the square root of the MSE, providing an error metric in the same units as the original data, making it easier to interpret.
R-squared (R^2 score): This metric tells us how well our model explains the variability of the output data. A value closer to 1 indicates a better fit.

19. Using these evaluation functions helps us understand how accurately our model is predicting the sales based on the input features.
20. In essence, evaluation functions help us determine whether the machine learning model's performance meets our expectations or requirements. This is the main goal of evaluating any model—to assess its effectiveness and make sure it's working as intended.
21. For the given dataset, the evaluation metrics are as follows:
Mean Squared Error (MSE): 2.22
Root Mean Squared Error (RMSE): 1.49
Mean Absolute Error (MAE): 1.21
22. These values indicate how well the model is performing in predicting sales based on the input features. Lower values generally suggest better model accuracy.

Evaluaton of Linear Regression

In [12]:
```
#Mean Absolute Error
#Mean Squared Error
#Root mean Squared Error
#R2 score

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
print(f"MSE is {mean_squared_error(y_test,y_pred_test)}")
print(f"RMSE is {mean_squared_error(y_test,y_pred_test, squared=False)}")
print(f"MAE is {mean_absolute_error(y_test,y_pred_test)}")
```

```
MSE is 2.2237746679115586
RMSE is 1.491232600204126
MAE is 1.211152512498049
```

23. If I calculate the R^2 score for this dataset, it is **0.91**. This means that using the features (TV, radio, and newspaper spending), I can predict the sales with **91% accuracy**. In other words, the model can explain **91% of the variance** in the output variable (sales) based on the input features. This indicates a strong relationship between the inputs and the target variable.

In [13]:
```
#R2 score
# How much variance can be explained by the given features
print(f"R2-score is {r2_score(y_test,y_pred_test)}")
```

```
R2-score is 0.912722072959617
```

24. To find the best values for the coefficients (often represented as θ0,θ1,θ2,…,θn\theta_0, \theta_1, \theta_2, \ldots, \theta_nθ0,θ1,θ2,…,θn) in a linear regression model, we need a method to measure how well the model is performing. This is where a **loss function** comes into play.
    - A loss function helps us quantify the difference between the predicted values and the actual values. It tells us how "off" our predictions are, allowing us to adjust the model to improve accuracy.
    - One of the most commonly used loss functions in linear regression is the **squared error loss**. Here's how it works:
    - **Calculating Errors**: For each data point, we calculate the error, which is the difference between the actual value and the predicted value.
    - **Squaring the Errors**: To ensure all errors are positive (since errors can be negative), we square each of these differences. This means larger errors are penalized more than smaller ones.
    - **Averaging the Squared Errors**: We then take the average of these squared differences across all data points. This average is what we refer to as the **Mean Squared Error (MSE)**.
25. The goal is to minimize this loss function during the training process. By adjusting the coefficients (θ\thetaθ) to minimize the squared error loss, we can improve the accuracy of our predictions. In summary, the loss function helps guide the model to find the best-fitting line by quantifying prediction errors.
26. If I calculate the R^2 score using lasso regression, it is **0.91122**. This means that even after applying lasso regression, the ability of the features (like TV, radio, and newspaper spending) to explain the target variable (sales) remains about the same.

In [14]:
```
# Regularization of Linear Regression
#Lasso Regularization (L1)
# Tends to make the coefficients to absolute zero
# add the absolute vale of magnitude of coefficient as penalty term to loss function
from sklearn.linear_model import Lasso
las = Lasso()
las.fit(X_train,y_train)
y_pred_test = las.predict(X_test)
print(f"R2-score is {r2_score(y_test,y_pred_test)}")
```

```
R2-score is 0.9112250903194143
```

In [15]:
```python
# Ridge Regularization (L2)
# Add squared magnitude of coefficients as penalty term for the loss funtion
# Result in never set the value of coefficients to absolute zero
from sklearn.linear_model import Ridge
ridge = Ridge()
ridge.fit(X_train,y_train)
y_pred_test = las.predict(X_test)
print(f"R2-score is {r2_score(y_test,y_pred_test)}")
```

```
R2-score is 0.9112250903194143
```

27. In addition to lasso and ridge regression, there is another regularization technique called elastic net regularization. This method combines both L1 (lasso) and L2 (ridge) regularization.
28. With elastic net, you can specify how much L1 regularization and how much L2 regularization you want to apply. This balance is controlled using a parameter. Essentially, elastic net allows for flexibility in applying both types of regularization to improve model performance and manage overfitting.

In [16]:
```python
# ElasticNet Regularization (L1 + L2)
# Add squared magnitude of coefficients as penalty term for the loss funtion
# Result in never set the value of coefficients to absolute zero
from sklearn.linear_model import ElasticNet
elasticnet = ElasticNet()
elasticnet.fit(X_train,y_train)
y_pred_test = elasticnet.predict(X_test)
print(f"R2-score is {r2_score(y_test,y_pred_test)}")
```

```
R2-score is 0.9117469302155123
```

29. Next, let's talk about the **assumptions of the linear regression algorithm**. When we apply linear regression, we need to make sure that the dataset we are using meets certain assumptions.
30. It is our responsibility to check that the data aligns with these assumptions because if they are not met, the results of the linear regression may not be reliable or accurate.
31. To check for a linear relationship, we can create a **pair plot**. This involves making scatter plots for each feature against the target variable.
32. By visualizing these plots, we can see if there is a linear relationship between the features (like TV, radio, and newspaper spending) and the target (sales). If the points in the scatter plots tend to form a straight line, it indicates a linear relationship.

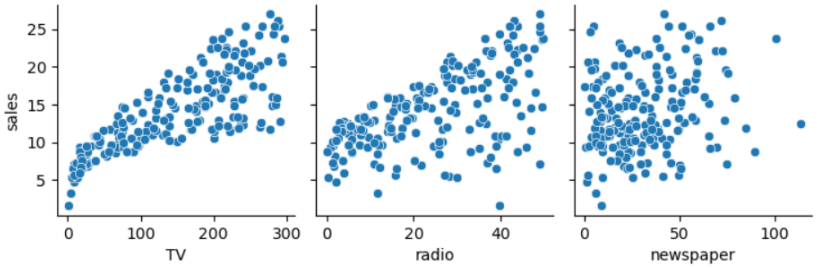
In [17]:
```
# Assumptions of Linear Regression
# Linear relationship between feature and target
df.head()
```

Out[17]:

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

In [18]:
```
import seaborn as sns
sns.pairplot(df,x_vars=['TV','radio','newspaper'],y_vars=['sales'])
plt.show()
```

```
C:\Users\PULKIT\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



33. In addition to visual methods, we can use a numerical method to assess relationships between variables. One way to do this is by calculating the **Pearson correlation coefficient**.
34. We can create a correlation matrix using the command `df.corr()`, which will show the correlation coefficients between all the variables in the dataset.
35. For instance, if we look specifically at the **sales** column, we might find that it has a correlation coefficient of **0.78** with **radio**. This indicates a strong positive relationship between radio spending and sales, meaning that as radio spending increases, sales tend to increase as well.

In [19]:
```
corr = df.corr()
corr
```

Out[19]:

	TV	radio	newspaper	sales
TV	1.000000	0.054809	0.056648	0.782224
radio	0.054809	1.000000	0.354104	0.576223
newspaper	0.056648	0.354104	1.000000	0.228299
sales	0.782224	0.576223	0.228299	1.000000

36. If I calculate the mean of the residuals, I find that it is **-0.075**, which is close to zero. This indicates that the mean residual is indeed near zero. Therefore, for this dataset, we can conclude that it satisfies the assumption of the linear regression algorithm.

```
In [20]: # mean residuals = 0
         residual = (y_test - y_pred_test)
         np.mean(residual)

Out[20]: -0.07559702903921801
```

37. Another assumption of linear regression is that the error terms are normally distributed. This means we expect the differences between the actual values and the predicted values (the residuals) to follow a normal distribution.
38. To check this assumption, we can use plots, such as a **histogram** or a **Q-Q plot** (quantile-quantile plot), to visualize the overall distribution of the residuals. These plots will help us see if the residuals approximate a normal distribution, which is important for the validity of the linear regression results.

```
In [21]: # Normal distribution of error terms
         sns.distplot(residual)
         plt.show()
```

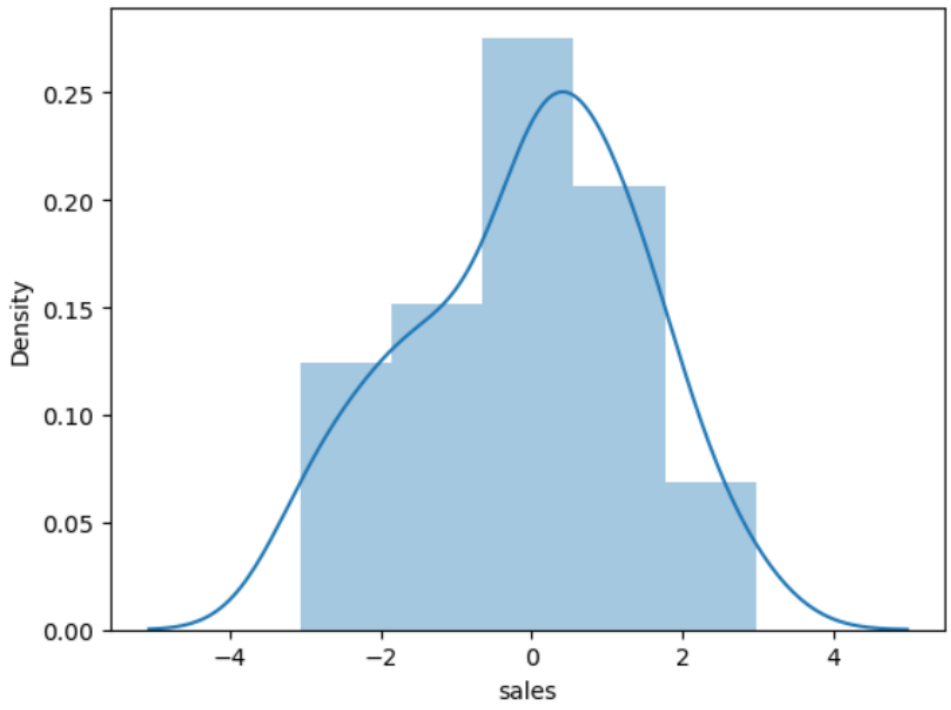
```
C:\Users\PULKIT\AppData\Local\Temp\ipykernel_25116\408193164.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(residual)
```



39. This code defines a function that calculates the VIF scores for the features in a dataset, helping to identify if there are any multicollinearity issues among the features. When you call this function with your feature data, it will return a table showing each feature along with its VIF score.

In [ ]:
```
# Multi collinearity
# Vif score
```

In [22]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [23]:
```
def vif_score(X):
    X_arr = X.values
    vif = [variance_inflation_factor(X_arr,i) for i in range(X.shape[1]) ]
    vif_score = pd.DataFrame()
    vif_score['vif_score'] = vif
    vif_score['Features'] = X.columns
    return vif_score
```

In [24]:
```
vif_score(X)
```

Out[24]:

	vif_score	Features
0	2.486772	TV
1	3.285462	radio
2	3.055245	newspaper