SUMMARY :

The objective of the task was to predict whether customers will make a purchase or not , we have been provided with the dataset with multiple columns , with no missing values .

we started with :

1. importing the train data , and by using **pandas** library we read the data and understood the data frame completely by using pd.nunique() , pd.head(), pd.describe() , pd.shape() and many more

Next we moved on

2. To visualizing the data by using **seaborn** and **matplot.pyplot** using these two libraries we were able to visualise the data more clearly

- we used countplot to understand the purchase distribution
- We used boxplot to detect outliers , boxplot is more suitable for outliers because it clearly marks points that lie far outside the normal range.

- correlation matrix : shows the **correlation coefficients** between all pairs of variables

  * This helps us to focus on features which are useful for predicting the outcome and can pick the most relevant predictors and improve our model's performance while simplifying it.

MOVING ON TO TRAINING THE ML MODEL

THE TWO ML MODELS THAT I HAVE CHOSEN TO TRAIN THE DATA :
1.LOGISTIC REGRESSION
2.XGBOOST

LOGISTIC REGRESSION :

1. WHY YOU THINK THE MODEL IS SUITABLE FOR THIS TASK

- since by the end of the task we need to predict whether the purchase is made , purchase which is the target variable is binary so it is perfect for us to use logistic regression
- logistic regression is simple and computationally fast to train .
- it not only just say yes or no it also gives probability between 0 and 1

2.THE MATHEMATICAL FORMULATION (KEY EQUATIONS, LOSS FUNCTIONS,)

MODEL EQUATION :

$$P(y=1|x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\cdots+\beta_n x_n)}}$$

- β_0 = intercept (bias)
- $\beta_1, \beta_2, \ldots, \beta_n$ = coefficients/weights for each feature

LOSS FUNCTION :

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

Where:

- y_i = actual label (0 or 1)
- $\hat{y}_i$ = predicted probability for class 1
- N = number of samples

3. ASSUMPTIONS BEHIND THE MODELS:

1. Linearity The log-odds (not probabilities) of the outcome are linear in features.
2. Independence: Features should not be highly correlated

3. No outliers: It’s sensitive to outliers unless preprocessed well.

4. HOW IT MAKES PREDICTIONS

in simple words, it uses the logic that if the person has more page_views, or cart_value or time_on site then it is more probable for them to make a purchase
Firstly it makes

1. Calculates a weighted sum of features like page views, time on site.
2. Applies a sigmoid to squash into a 0-1 range
3. Outputs a probability: like There’s a 75% chance this user will buy.
4. Applies a threshold (usually 0.5) to classify.

XGBOOST :

1. **WHY YOU THINK THE MODEL IS SUITABLE FOR THIS TASK**

XGBOOST s,is highly optimized implementation of the gradient boosting algorithm ,its an ensemble which combines multiple weak learners to form strong predictive model

- It Captures non-linear relationships e.g., maybe customers from Instagram only buy when cart value is high.
- Handles feature interactions automatically (like Time × Page Views).
- Typically outperforms simpler models like logistic regression in terms of accuracy.

2.THE MATHEMATICAL FORMULATION (KEY EQUATIONS, LOSS FUNCTIONS,)

MODEL EQUATIONS :

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

Where:

- f_k = the k-th decision tree
- $\mathcal{F}$ = space of regression trees
- $\hat{y}_i$ = final prediction after K trees

LOSS EQUATION :

$$l(y_i, \hat{y}_i) = -\left[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)\right]$$

But here, $\hat{y}_i$ is **updated iteratively** by boosting over decision trees.

3. ASSUMPTIONS BEHIND THE MODELS:

No strict assumptions about data distribution as it can handle non-linearity, missing values, and interactions between variables.

4. HOW IT MAKES PREDICTIONS

You can think of it as team , where each tree is trained not to be perfect , but to fix the mistake of the previous tree in the most efficient way possible

1. It starts with initial predictions which is usually the average value for regression or log odds for classification
2. It computes the error and find out how far the predictions are from the actual labels by gradient loss functions
3. It builds a shallow decision to predict where the model is going wrong
4. Add the predictions from this new tree to the current predictions

After Implementing and Training both models using libraries such as **scikit-learn**, **TensorFlow**, or **PyTorch ,** we evaluated

- Accuracy
- F1 Score
- Precision
- Recall

And these are the performance metric of both models

COMPARING BOTH THE MODEL BASED ON THEIR PERFORMANCE METRICS

	Model	Accuracy	F1 Score	Precision	Recall
0	Logistic Regression (Tuned)	0.766667	0.468354	0.616667	0.377551
1	XGBoost (Tuned)	0.788889	0.573034	0.637500	0.520408

COMPARATIVE ANALYSIS BETWEEN XGBOOST AND LOGISTIC REGRESSION

We can see from above that XGBOOST has performed better in every aspect

- This is because XGBOOST is advanced ensemble model and
- it capture feature interactions and non linearity ., whereas logistic regression cannot do both
- It also handles complex datasets easily better than logistic regression
- It has flexible decision boundaries as well

LOGISTIC REGRESSION :

STRENGTHS :

- It is simple , easy to interpret
- Fast to train
- Works well for binary target variable

WEAKNESS

- It struggles in real world messy data as it is complex and has non linear patterns

XGBOOST

STRENGTHS :

- It handles complex , high dimensional data
- It captures feature interactions automatically
- It works well with non linear data

WEAKNESS

- Slower to train

We can come into the conclusion that , **XGBoost** outperforms **Logistic Regression** in accuracy, F1 score, precision, and recall. This suggests that the dataset contains **non-linear patterns and interactions** that Logistic Regression cannot work on . While Logistic Regression is simpler and interpretable, XGBoost's tree-based approach makes it more powerful for complex classification tasks like this one. Therefore, **XGBoost is the preferred model** in this scenario due to its stronger ability to generalize , detect patterns and work with non linear patterns

Moving on we Performed Grid Search and Random Search to tune relevant hyperparameters for both models, this is important because it will Give us the **best parameters** and make the model more

- Smarter
- Efficient
- Boosts improvement
- It helps us to prevent overfitting or underfitting

These are the evaluation metrics **before tuning**

	Model	Accuracy	F1 Score	Precision	Recall
0	Logistic Regression	0.766667	0.468354	0.616667	0.377551
1	XGBoost	0.755556	0.516484	0.559524	0.479592

and these are the evaluation metrics **after tuning**

	Model	Accuracy	F1 Score	Precision	Recall
0	Logistic Regression (Tuned)	0.766667	0.468354	0.616667	0.377551
1	XGBoost (Tuned)	0.788889	0.573034	0.637500	0.520408

We can see there is a good amount of differences in **XGBOOST**
There is no much change in **logistic regression** because it is a simple model and it has very few hyperparameters to depend on

Key insights and learnings : From this task I understood how to tackle real -world problems with the ML model . How , when and why to implement the particular ML algorithm , mathematics involved behind it and the logic used. I also came to know about the differences of different models and why one is better than the other from the evaluation metrics .