**MACHINE LEARNIING**

**LAB MANUAL**

**B.TECH VI SEMESTER**

**(R22 REGULATION)**

**COMPUTER SCIENCE AND ENGINEERING**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**VAAGDEVI COLLEGE OF ENGINEERING**

**BOLLIKUNTA, WARANGAL**

**SYLLABUS FOR MACHINE LEARNING LAB**

**List of Experiments**

1. Write a python program to compute Central Tendency Measures: Mean, Median, Mode Measure of Dispersion: Variance, Standard Deviation?

2. Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy.

3. Study of Python Libraries for ML application such as Pandas and Matplotlib.

4. Write a Python program to implement Simple Linear Regression.

5. Implementation of Multiple Linear Regression for House Price Prediction using sklearn.

6. Implementation of Decision tree using sklearn and its parameter tuning.

7. Implementation of KNN using sklearn.

8. Implementation of Logistic Regression using sklearn.

9. Implementation of K-Means Clustering.

10. Performance analysis of Classification Algorithms on a specific dataset (Mini Project)

**Course Objective:**

* The objective of this lab is to get an overview of the various machine learning

techniques andcan demonstrate them using python.

**Course Outcomes:**

1. Understand modern notions in predictive data analysis.

2. Select data, model selection, model complexity and identify the trends.

3. Understand a range of machine learning algorithms along with their strengths andweaknesses

4. Build predictive models from data and analyze their performance.

**TEXT BOOK:**

1. Machine Learning – Tom M. Mitchell, - MGH.

**REFERENCE BOOK:**

1. Machine Learning: An Algorithmic Perspective, Stephen Marshland, Taylor & Francis.

List of Experiments

1. Write a python program to compute Central Tendency Measures:

**WEEK-1:**

1. write a python program to compute central tendency measures mean median mode measure of dispersion variance standard deviation

import statistics

def calculate\_mean(data):

return sum(data) / len(data)

def calculate\_median(data):

sorted\_data = sorted(data)

n = len(sorted\_data)

if n % 2 == 0:

middle1 = sorted\_data[n // 2 - 1]

middle2 = sorted\_data[n // 2]

return (middle1 + middle2) / 2

else:

return sorted\_data[n // 2]

def calculate\_mode(data):

returnstatistics.mode(data)

def calculate\_variance(data):

mean\_value = calculate\_mean(data)

squared\_diff\_sum = sum((x - mean\_value) \*\* 2 for x in data)

return squared\_diff\_sum / (len(data) - 1)

def calculate\_standard\_deviation(data):

variance\_value = calculate\_variance(data)

return variance\_value \*\* 0.5

# Example dataset

dataset = [10, 20, 30, 40, 50]

mean\_value = calculate\_mean(dataset)

median\_value = calculate\_median(dataset)

mode\_value = calculate\_mode(dataset)

variance\_value = calculate\_variance(dataset)

std\_deviation\_value = calculate\_standard\_deviation(dataset)

print(f"Dataset: {dataset}")

print(f"Mean: {mean\_value:.2f}")

print(f"Median: {median\_value:.2f}")

print(f"Mode: {mode\_value}")

print(f"Variance: {variance\_value:.2f}")

print(f"Standard Deviation: {std\_deviation\_value:.2f}")

output : Dataset: [10, 20, 30, 40, 50]

Mean: 30.00

Median: 30.00

Mode: 10

Variance: 250.00

Standard Deviation: 15.

**OUTPUT** : Dataset: [10, 20, 30, 40, 50]

Mean: 30.00

Median: 30.00

Mode: 10

Variance: 250.00

Standard Deviation: 15.81

**WEEK-2:**

2) Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy

Import numpy as np

# Create a 1D array

arr1d = np.array([1, 2, 3])

print(f"1D Array: {arr1d}")

# Create a 2D

arrayarr2d = np.array([[1, 2, 3], [4, 5, 6]])

print(f"2D Array: {arr2d}").

=> SciPy(Scientific Python):SciPy builds upon NumPy and provides additional functionality for scientific and engineering purposes.

It includes modules for optimization, integration, linear algebra, signal processing, and more.

Example usage:Pythonimportscipy.optimize as opt

# Solve an optimization problem

def objective(x):

return x[0]\*\*2 + x[1]\*\*2

result = opt.minimize(objective, [1, 1])

print(f"Optimal solution: {result.x}")

# Perform numerical integration

fromscipy.integrate import quad

def integrand(x):

return x\*\*2

area, error = quad(integrand, 0, 2)

print(f"Integral result: {area:.2f}")

1. Python’s Built-instatisticsModule: Thestatisticsmodule,
2. introduced in Python 3.4, provides functions for calculating mathematical statistics of numeric data. It includes measures of central location (such as mean, median, and mode) and measures of spread (such as standard deviation and variance).
3. Note that this module is not intended to compete with third-party libraries like NumPy or SciPy, which offer more extensive statistical capabilities.
4. Instead, it’s suitable for basic statistical calculations and graphing Customization: You can control colors, labels, and other plot properties.

**WEEK-3:**

3) Study of Python Libraries for ML application such as Pandas and Matplotlib.

Integration with Pandas: Matplotlib works seamlessly with Pandas

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df=pd.read\_csv('CAR DETAILS FROM CAR DEKHO.csv')

df

df['fuel'].value\_counts()

df['owner'].value\_counts()

df['seller\_type'].value\_counts()

df['transmission'].value\_counts()

le=LabelEncoder()

df['fuel']=le.fit\_transform(df['fuel'])

df['seller\_type']=le.fit\_transform(df['seller\_type'])

df['transmission']=le.fit\_transform(df['transmission'])

df['owner']=le.fit\_transform(df['owner'])

df

plt.figure(figsize=(12,6))

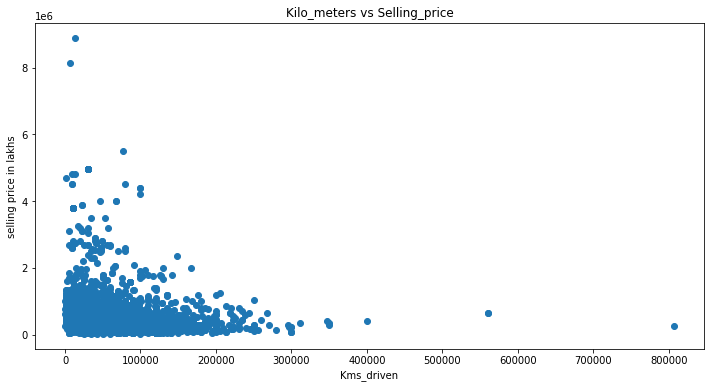
plt.scatter(x=df['km\_driven'],y=df['selling\_price'])

plt.xlabel('Kms\_driven')

plt.ylabel('selling price in lakhs')

plt.title('Kilo\_meters vs Selling\_price')

plt.show()



plt.figure(figsize=(12,6))

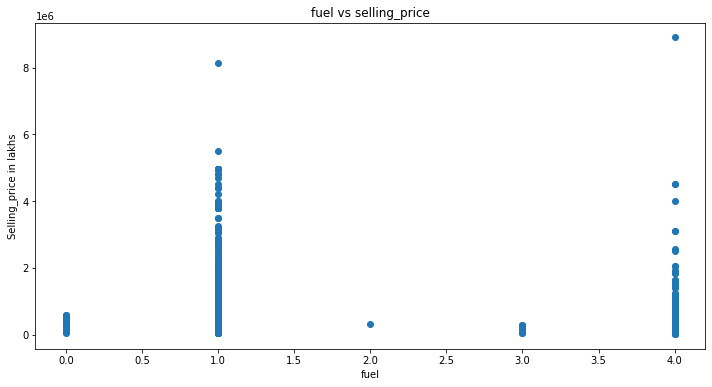
plt.scatter(x=df['fuel'],y=df['selling\_price'])

plt.xlabel('fuel')

plt.ylabel('Selling\_price in lakhs')

plt.title('fuel vs selling\_price')

plt.show()



plt.figure(figsize=(12,6))

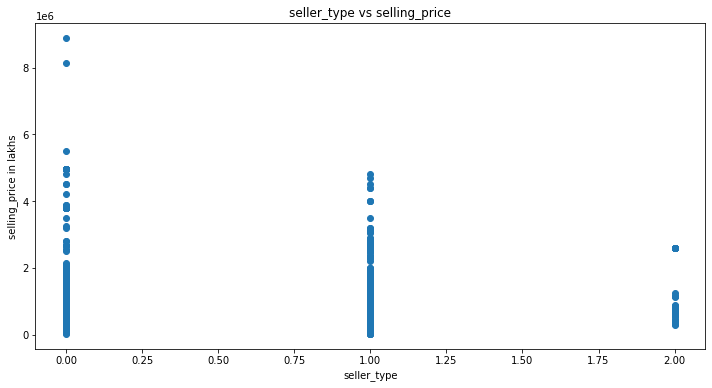
plt.scatter(x=df['seller\_type'],y=df['selling\_price'])

plt.xlabel('seller\_type')

plt.ylabel('selling\_price in lakhs')

plt.title('seller\_type vs selling\_price')

plt.show()



plt.scatter(x=df['transmission'],y=df['selling\_price'])

plt.xlabel('transmission')

plt.ylabel('selling\_price in lakhs')

plt.title('Transmission vs Selling\_price')

plt.show()



plt.figure(figsize=(15,9))

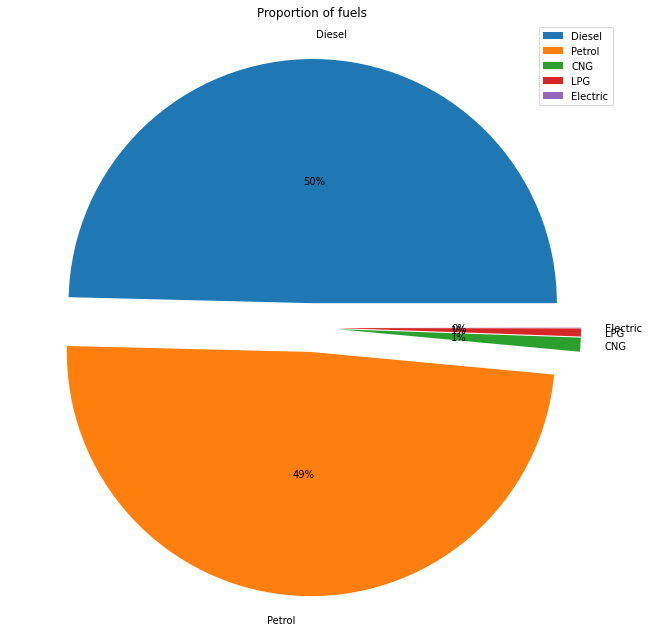
plt.pie(x=df['fuel'].value\_counts(),autopct='%1.0f%%', pctdistance=0.5,explode=[0.1,0.1,0.1,0.1,0.1],labels=['Diesel','Petrol','CNG','LPG','Electric'])

plt.title('Proportion of fuels')

plt.legend()

plt.tight\_layout()

plt.show()



plt.figure(figsize=(15,9))

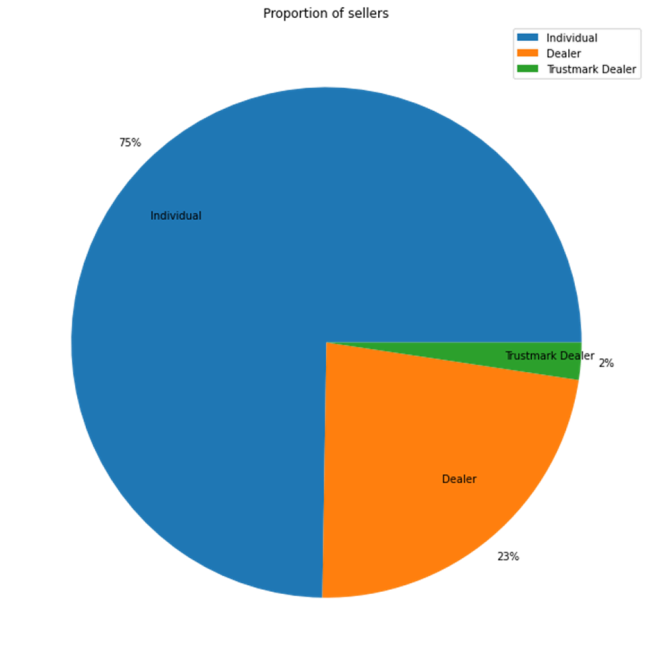
plt.pie(x=df['seller\_type'].value\_counts(),autopct='%1.0f%%', pctdistance=1.1, labeldistance=0.7,labels=['Individual','Dealer','Trustmark Dealer'])

plt.title('Proportion of sellers')

plt.legend()

plt.tight\_layout()

plt.show()



**WEEK-4:**

4) Write a python program to implement simple linear regression

# Import necessary libraries

Importnumpy as np

fromsklearn.linear\_model import LinearRegression

importmatplotlib.pyplot as plt

# Generate some sample data

X = np.array([[1], [2], [3], [4], [5]]) # Independent variable (feature)

y = np.array([2, 4, 5, 4, 6]) # Dependent variable (response)

# Create a linear regression model

reg = LinearRegression().fit(X, y)

# Get the coefficients and intercept

slope = reg.coef\_[0]

intercept = reg.intercept\_

print(f"Slope (Coefficient): {slope:.2f}")

print(f"Intercept: {intercept:.2f}")

# Predict a new value

new\_X = np.array([[6]])

predicted\_y = reg.predict(new\_X)

print(f"Predicted value for X = 6: {predicted\_y[0]:.2f}")

# Plot the data and regression line

plt.scatter(X, y, color='blue', label='Data points')

plt.plot(X, reg.predict(X), color='red', label='Regression line')

plt.xlabel('X')

plt.ylabel('y')

plt.title('Simple Linear Regression')

plt.legend()

plt.show()

**OUTPUT:**

Slope (Coefficient): 0.80

Intercept: 1.60

Predicted value for X = 6: 6.40

**WEEK-5:**

5) Implementation of multiple linear regression for house price prediction using sklearn.

|  |
| --- |
| # importing modules and packages  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import seaborn as sns  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error  from sklearn import preprocessing    # importing data  df = pd.read\_csv('Real estate.csv')  df.drop('No', inplace=True, axis=1)    print(df.head())    print(df.columns)    # plotting a scatterplot  sns.scatterplot(x='X4 number of convenience stores',                  y='Y house price of unit area', data=df)    # creating feature variables  X = df.drop('Y house price of unit area', axis=1)  y = df['Y house price of unit area']    print(X)  print(y)    # creating train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(      X, y, test\_size=0.3, random\_state=101)    # creating a regression model  model = LinearRegression()    # fitting the model  model.fit(X\_train, y\_train)    # making predictions  predictions = model.predict(X\_test)    # model evaluation  print('mean\_squared\_error : ', mean\_squared\_error(y\_test, predictions))  print('mean\_absolute\_error : ', mean\_absolute\_error(y\_test, predictions)) |

**OUTPUT:**

   X1 transaction date  X2 house age  …  X6 longitude  Y house price of unit area

0             2012.917          32.0  …     121.54024                        37.9

1             2012.917          19.5  …     121.53951                        42.2

2             2013.583          13.3  …     121.54391                        47.3

3             2013.500          13.3  …     121.54391                        54.8

4             2012.833           5.0  …     121.54245                        43.1

[5 rows x 7 columns]

Index([‘X1 transaction date’, ‘X2 house age’,

      ‘X3 distance to the nearest MRT station’,

      ‘X4 number of convenience stores’, ‘X5 latitude’, ‘X6 longitude’,

      ‘Y house price of unit area’],

     dtype=’object’)

    X1 transaction date  X2 house age  …  X5 latitude  X6 longitude

0               2012.917          32.0  …     24.98298     121.54024

1               2012.917          19.5  …     24.98034     121.53951

2               2013.583          13.3  …     24.98746     121.54391

3               2013.500          13.3  …     24.98746     121.54391

4               2012.833           5.0  …     24.97937     121.54245

..                   …           …  …          …           …

409             2013.000          13.7  …     24.94155     121.50381

410             2012.667           5.6  …     24.97433     121.54310

411             2013.250          18.8  …     24.97923     121.53986

412             2013.000           8.1  …     24.96674     121.54067

413             2013.500           6.5  …     24.97433     121.54310

[414 rows x 6 columns]

0      37.9

1      42.2

2      47.3

3      54.8

4      43.1

      …

409    15.4

410    50.0

411    40.6

412    52.5

413    63.9

Name: Y house price of unit area, Length: 414, dtype: float64

mean\_squared\_error :  46.21179783493418

mean\_absolute\_error :  5.392293684756571

**WEEK-6:**

6) Implementation of Decision tree using sklearn and its parameter tuning.

# Importing the required packages

import numpy as np

import pandas as pd

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

# Function to import the dataset

def importdata():

balance\_data = pd.read\_csv(

‘/ml/machine-learning-' +

'databases/balance-scale/balance-scale.data',

sep=',', header=None)

# Displaying dataset information

print("Dataset Length: ", len(balance\_data))

print("Dataset Shape: ", balance\_data.shape)

print("Dataset: ", balance\_data.head())

return balance\_data

# Function to split the dataset into features and target variables

def splitdataset(balance\_data):

# Separating the target variable

X = balance\_data.values[:, 1:5]

Y = balance\_data.values[:, 0]

# Splitting the dataset into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size=0.3, random\_state=100)

return X, Y, X\_train, X\_test, y\_train, y\_test

def train\_using\_gini(X\_train, X\_test, y\_train):

# Creating the classifier object

clf\_gini = DecisionTreeClassifier(criterion="gini",

random\_state=100, max\_depth=3, min\_samples\_leaf=5)

# Performing training

clf\_gini.fit(X\_train, y\_train)

return clf\_gini

def train\_using\_entropy(X\_train, X\_test, y\_train):

# Decision tree with entropy

clf\_entropy = DecisionTreeClassifier(

criterion="entropy", random\_state=100,

max\_depth=3, min\_samples\_leaf=5)

# Performing training

clf\_entropy.fit(X\_train, y\_train)

return clf\_entropy

# Function to make predictions

def prediction(X\_test, clf\_object):

y\_pred = clf\_object.predict(X\_test)

print("Predicted values:")

print(y\_pred)

return y\_pred

# Placeholder function for cal\_accuracy

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",

confusion\_matrix(y\_test, y\_pred))

print("Accuracy : ",

accuracy\_score(y\_test, y\_pred)\*100)

print("Report : ",

classification\_report(y\_test, y\_pred))

# Function to plot the decision tree

def plot\_decision\_tree(clf\_object, feature\_names, class\_names):

plt.figure(figsize=(15, 10))

plot\_tree(clf\_object, filled=True, feature\_names=feature\_names, class\_names=class\_names, rounded=True)

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)

clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)

clf\_entropy = train\_using\_entropy(X\_train, X\_test, y\_train)

# Visualizing the Decision Trees

plot\_decision\_tree(clf\_gini, ['X1', 'X2', 'X3', 'X4'], ['L', 'B', 'R'])

plot\_decision\_tree(clf\_entropy, ['X1', 'X2', 'X3', 'X4'], ['L', 'B', 'R'])

**OUTPUT:**

**DATA INFO**

**Dataset Length:** 625

**Dataset Shape:** (625, 5)

**Dataset:** 0 1 2 3 4

0 B 1 1 1 1

1 R 1 1 1 2

2 R 1 1 1 3

3 R 1 1 1 4

4 R 1 1 1 5

**WEEK-7:**

7) Implementation of KNN using sklearn.

*#Load the necessary python libraries*

importnumpyasnp

importpandasaspd

importmatplotlib.pyplotasplt

plt.style.use('ggplot')

*#Load the dataset*

df=pd.read\_csv('../input/diabetes.csv')

*#Print the first 5 rows of the dataframe.*

df.head()

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI DiabetesPedigreeFunction Age Outcome

0 6 148 72 35 0 33.6 0.627 50 1

1 1 85 66 29 0 26.6 0.351 31 0

2 8 183 64 0 0 23.3 0.672 32 1

3 1 89 66 23 94 28.1 0.167 21 0

4 0 137 40 35 168 43.1 2.288 33 1

*#Let's observe the shape of the dataframe.*

df.shape

Out[3]:

(768, 9)

*#Let's create numpy arrays for features and target*

X=df.drop('Outcome',axis=1).values

y=df['Outcome'].values

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.4,random\_state=42,stratify=y)

*#import KNeighborsClassifier*

from sklearn.neighbors import KNeighborsClassifier

*#Setup arrays to store training and test accuracies*

neighbors= np.arange(1,9)

train\_accuracy =np.empty(len(neighbors))

test\_accuracy =np.empty(len(neighbors))

for i,k **in**enumerate(neighbors):

*#Setup a knn classifier with k neighbors*

knn= KNeighborsClassifier(n\_neighbors=k)

*#Fit the model*

knn.fit(X\_train, y\_train)

*#Compute accuracy on the training set*

train\_accuracy[i] =knn.score(X\_train, y\_train)

*#Compute accuracy on the test set*

test\_accuracy[i] =knn.score(X\_test, y\_test)

In [8]:

*#Generate plot*

plt.title('k-NN Varying number of neighbors')

plt.plot(neighbors, test\_accuracy, label='Testing Accuracy')

plt.plot(neighbors, train\_accuracy, label='Training accuracy')

plt.legend()

plt.xlabel('Number of neighbors')

plt.ylabel('Accuracy')

plt.show()

We can observe above that we get maximum testing accuracy for k=7. So lets create a KNeighborsClassifier with number of neighbors as 7.

In [9]:

*#Setup a knn classifier with k neighbors*

knn= KNeighborsClassifier(n\_neighbors=7)

In [10]:

*#Fit the model*

knn.fit(X\_train,y\_train)

Out[10]:

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=7, p=2,

weights='uniform')

#In [11]:

linkcode

*#Get accuracy. Note: In case of classification algorithms score method represents accuracy.*

knn.score(X\_test,y\_test)

#Out[11]:

0.7305194805194806

#import confusion\_matrix

from sklearn.metrics import confusion\_matrix

#let us get the predictions using the classifier we had fit above

y\_pred = knn.predict(X\_test)

confusion\_matrix(y\_test,y\_pred)

array([[165, 36],

[ 47, 60]])

#Considering confusion matrix above:

True negative = 165

False positive = 36

True postive = 60

Fasle negative = 47

#Confusion matrix can also be obtained using crosstab method of pandas.

pd.crosstab(y\_test, y\_pred, rownames=['True'], colnames=['Predicted'], margins=True)

Predicted 0 1 All

True

0 165 36 201

1 47 60 107

All 212 96 308

**WEEK-8:**

8) Implementation of Logistic Regression using sklearn.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read\_csv('bank.csv')

df

df.head()

df.tail()

df.info()

df.describe()

df.isnull().sum()

df['loan'].value\_counts()

df['deposit'].unique()

plt.figure(figsize=(17,6))

sns.countplot('age', hue = 'deposit', data = df)

plt.figure(figsize = (17, 6))

sns.countplot('education', hue = 'deposit', data = df)

plt.figure(figsize = (17, 6))

sns.countplot('job', hue = 'deposit', data = df)

df.shape

df.drop(columns=['contact','day','month','duration','campaign','pdays','previous','poutcome'],inplace=True)

df

df['job'].unique()

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

df['job']=le.fit\_transform(df['job'])

df['job']

df['marital']=le.fit\_transform(df['marital'])

df['education']=le.fit\_transform(df['education'])

df['default']=le.fit\_transform(df['default'])

df['housing']=le.fit\_transform(df['housing'])

df['deposit']=le.fit\_transform(df['deposit'])

df['loan']=le.fit\_transform(df['loan'])

df

X=df.iloc[:,0:-1].values

y=df.iloc[:,-1].values

X

y

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)

X\_train

X\_test

y\_train

y\_test

from sklearn.linear\_model import LogisticRegression

lr=LogisticRegression()

y\_pred=lr.fit(X\_train,y\_train)

y\_predict=lr.predict(X\_test)

y\_predict

o/p: array([0, 0, 0, ..., 1, 0, 1])

print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(lr.score(X\_test, y\_test)))

**OUTPUT:**

Accuracy of logistic regression classifier on test set: 0.60

**WEEK-9:**

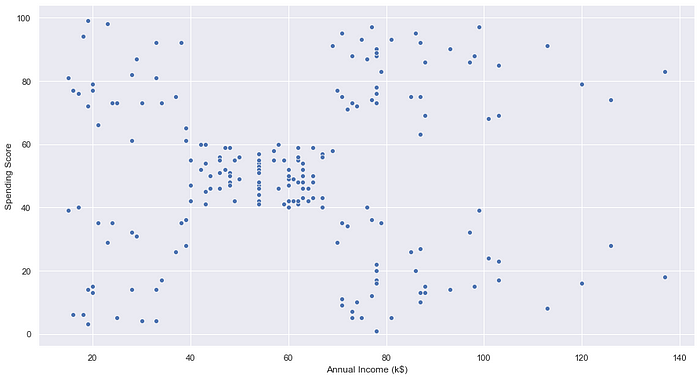
9) Implementation of K-Means Clustering.

import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.cluster import Kmeans

customer\_data = pd.read\_csv('Customers.csv')  
customer\_data.head()  
customer\_data.info()  
customer\_data.isnull().sum()  
X = customer\_data.drop(columns=['CustomerID','Gender','Age'], axis=1).values

## #Visualize the data points

plt.figure(figsize=(15,8))  
sns.scatterplot(X[:,0], X[:, 1])  
plt.xlabel('Annual Income (k$)')  
plt.ylabel('Spending Score')  
plt.show()



## #Find the K value using the Elbow method

wcss=[]  
for i in range(1,11):  
 kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=2)  
 kmeans.fit(X)  
 wcss.append(kmeans.inertia\_)  
plt.figure(figsize=(15,8))  
plt.plot(range(1,11), wcss)  
plt.title('The Elbow Point Graph')  
plt.xlabel('Number of Clusters (K)')  
plt.ylabel('wcss')  
plt.show()

#WCSS doesn’t reduce much after k=5. So, we can choose 5 as the perfect K value or Clusters.

## #Training the K-means algorithm on the training dataset

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state=0)  
Y = kmeans.fit\_predict(X)

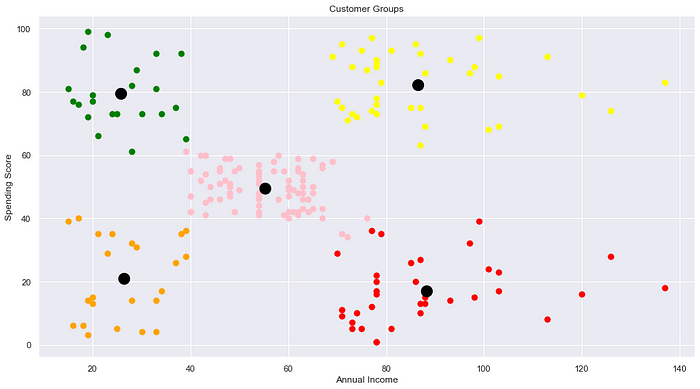
## #Centroid points

kmeans.cluster\_centers\_

array([[88.2 , 17.11428571],  
[55.2962963 , 49.51851852],  
[86.53846154, 82.12820513],  
[25.72727273, 79.36363636],  
[26.30434783, 20.91304348]])

#Visualize the clusters formed

plt.figure(figsize=(15,8))  
plt.scatter(X[Y==0,0], X[Y==0,1], s=50, c='red', label='Cluster 1')  
plt.scatter(X[Y==1,0], X[Y==1,1], s=50, c='pink', label='Cluster 2')  
plt.scatter(X[Y==2,0], X[Y==2,1], s=50, c='yellow', label='Cluster 3')  
plt.scatter(X[Y==3,0], X[Y==3,1], s=50, c='green', label='Cluster 4')  
plt.scatter(X[Y==4,0], X[Y==4,1], s=50, c='orange', label='Cluster 5')  
plt.scatter(kmeans.cluster\_centers\_[:,0], kmeans.cluster\_centers\_[:,1], s=200, c='black', label='Centroids')  
plt.title('Customer Groups')  
plt.xlabel('Annual Income')  
plt.ylabel('Spending Score')  
plt.show()



## **Conclusion:**

In this blog, we discussed one of the most famous clustering algorithms: K-Means. We can clearly see the five clusters formed. The black dots represent the centroid for each cluster.

**WEEK-10:**

10) Performance analysis of Classification Algorithms on a specific dataset (Mini

Project)

# 💳💲 Loan Prediction with Various ML Models 💵

**Table of Contents**

1. Introduction

2. Importing Libraries

3. Reading Data Set

4. Data Exploration

5. Data Preprocessing

6. Models

7. Model Comparison

# **1. Introduction**



## **Data Set Problems**

👉 The company seeks **to automate (in real time) the loan qualifying procedure** based on information given by customers while filling out an online application form. It is expected that the development of ML models that can help the company predict loan approval in **accelerating decision-making process** for determining whether an applicant is eligible for a loan or not.

## **Objectives of Project**

👉 **This project aims to:**

* Analyze customer data provided in data set (EDA)
* Build various ML models that can predict loan approval

‍💻 **The machine learning models used in this project are:**

1. Logistic Regression
2. K-Nearest Neighbour (KNN)
3. Support Vector Machine (SVM)
4. Naive Bayes
5. Decision Tree
6. Random Forest
7. Gradient Boost

## **Data Set Description**

👉There are **13 variables** in this data set:

* **8 categorical** variables,
* **4 continuous** variables, and
* **1** variable to accommodate the loan ID.

👉The following is the **structure of the data set**.

| **Variable Name** | **Description** | **Sample Data** |
| --- | --- | --- |
| **Loan\_ID** | Loan reference number (unique ID) | LP001002; LP001003; ... |
| **Gender** | Applicant gender (Male or Female) | Male; Female |
| **Married** | Applicant marital status (Married or not married) | Married; Not Married |
| **Dependents** | Number of family members | 0; 1; 2; 3+ |
| **Education** | Applicant education/qualification (graduate or not graduate) | Graduate; Under Graduate |
| **Self\_Employed** | Applicant employment status (yes for self-employed, no for employed/others) | Yes; No |
| **ApplicantIncome** | Applicant's monthly salary/income | 5849; 4583; ... |
| **CoapplicantIncome** | Additional applicant's monthly salary/income | 1508; 2358; ... |
| **LoanAmount** | Loan amount | 128; 66; ... |
| **Loan\_Amount\_Term** | The loan's repayment period (in days) | 360; 120; ... |
| **Credit\_History** | Records of previous credit history (0: bad credit history, 1: good credit history) | 0; 1 |
| **Property\_Area** | The location of property (Rural/Semiurban/Urban) | Rural; Semiurban; Urban |
| **Loan\_Status** | Status of loan (Y: accepted, N: not accepted) | Y; N |

# **2. Importing Libraries**

👉Importing libraries that will be used in this notebook.

In [1]:

Import numpyasnp

Import pandasaspd

Import matplotlib.pyplotasplt

Import missingnoasmso

Import seabornassns

Import warnings

Import os

Import scipy

From scipy import stats

From scipy.stats import pearsonr

From scipy.stats importttest\_ind

From sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from imblearn.over\_sampling import SMOTE

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import CategoricalNB

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

from xgboost import XGBClassifier

from sklearn.model\_selection import GridSearchCV,RandomizedSearchCV

# **3. Reading Data Set**

👉After importing libraries, we will also import the dataset that will be used.

In [2]:

df=pd.read\_csv("../input/loan-predication.csv")

df.head()

df.shape

(614, 13)

As can be seen, the **13 columns** are readable. It also can be seen that there are **614 observations** in the data set.

# **4. Data Exploration**

👉This section will perform data exploration of "raw" data set that has been imported.

## **4.1 Categorical Variable**

👉The first type of variable that I will explore is categorical variable.

### 4.1.1 Loan ID

In [4]:

df.Loan\_ID.value\_counts(dropna=False)

Out[4]:

LP001002 1

LP002328 1

LP002305 1

LP002308 1

LP002314 1

..

LP001692 1

LP001693 1

LP001698 1

LP001699 1

LP002990 1

Name: Loan\_ID, Length: 614, dtype: int64

👉 It can be seen that there are 614 unique ID in the dataset.

### 4.1.2 Gender

In [5]:

df.Gender.value\_counts(dropna=False)

Out[5]:

Male 489

Female 112

NaN 13

Name: Gender, dtype: int64

In [6]:

sns.countplot(x="Gender",data=df,palette="hls")

plt.show()

In [7]:

countMale=len(df[df.Gender=='Male'])

countFemale=len(df[df.Gender=='Female'])

countNull=len(df[df.Gender.isnull()])

print("Percentage of Male applicant: **{:.2f}**%".format((countMale/(len(df.Gender))\*100)))

print("Percentage of Female applicant: **{:.2f}**%".format((countFemale/(len(df.Gender))\*100)))

print("Missing values percentage: **{:.2f}**%".format((countNull/(len(df.Gender))\*100)))

Percentage of Male applicant: 79.64%

Percentage of Female applicant: 18.24%

Missing values percentage: 2.12%

👉From the results above, the number of male applicants is higher compared to female applicants. It also can be seen there are missing values in this column.

### 4.1.3 Married

In [8]:

df.Married.value\_counts(dropna=False)

Out[8]:

Yes 398

No 213

NaN 3

Name: Married, dtype: int64

In [9]:

sns.countplot(x="Married",data=df,palette="Paired")

plt.show()

👉The number of applicants that has been married is higher compared to applicants that hasn't married. It also can be seen there are small number of missing values in this column.

In [10]:

countMarried=len(df[df.Married=='Yes'])

countNotMarried=len(df[df.Married=='No'])

countNull=len(df[df.Married.isnull()])

print("Percentage of married: **{:.2f}**%".format((countMarried/(len(df.Married))\*100)))

print("Percentage of Not married applicant: **{:.2f}**%".format((countNotMarried/(len(df.Married))\*100)))

print("Missing values percentage: **{:.2f}**%".format((countNull/(len(df.Married))\*100)))

Percentage of married: 64.82%

Percentage of Not married applicant: 34.69%

Missing values percentage: 0.49%

### 4.1.4 Education

In [11]:

df.Education.value\_counts(dropna=False)

Out[11]:

Graduate 480

Not Graduate 134

Name: Education, dtype: int64

In [12]:

sns.countplot(x="Education",data=df,palette="rocket")

plt.show()

In [13]:

countGraduate=len(df[df.Education=='Graduate'])

countNotGraduate=len(df[df.Education=='Not Graduate'])

countNull=len(df[df.Education.isnull()])

print("Percentage of graduate applicant: **{:.2f}**%".format((countGraduate/(len(df.Education))\*100)))

print("Percentage of Not graduate applicant: **{:.2f}**%".format((countNotGraduate/(len(df.Education))\*100)))

print("Missing values percentage: **{:.2f}**%".format((countNull/(len(df.Education))\*100)))

Percentage of graduate applicant: 78.18%

Percentage of Not graduate applicant: 21.82%

Missing values percentage: 0.00%

👉The number of applicants that has been graduated is higher compared to applicants that hasn't graduated.

### 4.1.5 Self Employed

In [14]:

df.Self\_Employed.value\_counts(dropna=False)

Out[14]:

No 500

Yes 82

NaN 32

Name: Self\_Employed, dtype: int64

In [15]:

sns.countplot(x="Self\_Employed",data=df,palette="crest")

plt.show()

In [16]:

countNo=len(df[df.Self\_Employed=='No'])

countYes=len(df[df.Self\_Employed=='Yes'])

countNull=len(df[df.Self\_Employed.isnull()])

print("Percentage of Not self employed: **{:.2f}**%".format((countNo/(len(df.Self\_Employed))\*100)))

print("Percentage of self employed: **{:.2f}**%".format((countYes/(len(df.Self\_Employed))\*100)))

print("Missing values percentage: **{:.2f}**%".format((countNull/(len(df.Self\_Employed))\*100)))

Percentage of Not self employed: 81.43%

Percentage of self employed: 13.36%

Missing values percentage: 5.21%

👉The number of applicants that are not self employed is higher compared to applicants that are self employed. It also can be seen, there are missing values in this column.

### 4.1.6 Credit History

In [17]:

df.Credit\_History.value\_counts(dropna=False)

Out[17]:

1.0 475

0.0 89

NaN 50

Name: Credit\_History, dtype: int64

In [18]:

sns.countplot(x="Credit\_History",data=df,palette="viridis")

plt.show()

In [19]:

count1=len(df[df.Credit\_History==1])

count0=len(df[df.Credit\_History==0])

countNull=len(df[df.Credit\_History.isnull()])

print("Percentage of Good credit history: **{:.2f}**%".format((count1/(len(df.Credit\_History))\*100)))

print("Percentage of Bad credit history: **{:.2f}**%".format((count0/(len(df.Credit\_History))\*100)))

print("Missing values percentage: **{:.2f}**%".format((countNull/(len(df.Credit\_History))\*100)))

Percentage of Good credit history: 77.36%

Percentage of Bad credit history: 14.50%

Missing values percentage: 8.14%

👉The number of applicants that have good credit history is higher compared to applicants that have bad credit history. It also can be seen, there are missing values in this column.

### 4.1.7 Property Area

In [20]:

df.Property\_Area.value\_counts(dropna=False)

Out[20]:

Semiurban 233

Urban 202

Rural 179

Name: Property\_Area, dtype: int64

In [21]:

sns.countplot(x="Property\_Area",data=df,palette="cubehelix")

plt.show()

In [22]:

countUrban=len(df[df.Property\_Area=='Urban'])

countRural=len(df[df.Property\_Area=='Rural'])

countSemiurban=len(df[df.Property\_Area=='Semiurban'])

countNull=len(df[df.Property\_Area.isnull()])

print("Percentage of Urban: **{:.2f}**%".format((countUrban/(len(df.Property\_Area))\*100)))

print("Percentage of Rural: **{:.2f}**%".format((countRural/(len(df.Property\_Area))\*100)))

print("Percentage of Semiurban: **{:.2f}**%".format((countSemiurban/(len(df.Property\_Area))\*100)))

print("Missing values percentage: **{:.2f}**%".format((countNull/(len(df.Property\_Area))\*100)))

Percentage of Urban: 32.90%

Percentage of Rural: 29.15%

Percentage of Semiurban: 37.95%

Missing values percentage: 0.00%

👉This column has a balanced distribution between Urban, Rural, and Semiurban property area. It also can be seen there is no missing value.

### 4.1.8 Loan Status

In [23]:

df.Loan\_Status.value\_counts(dropna=False)

Out[23]:

Y 422

N 192

Name: Loan\_Status, dtype: int64

In [24]:

sns.countplot(x="Loan\_Status",data=df,palette="YlOrBr")

plt.show()

In [25]:

countY=len(df[df.Loan\_Status=='Y'])

countN=len(df[df.Loan\_Status=='N'])

countNull=len(df[df.Loan\_Status.isnull()])

print("Percentage of Approved: **{:.2f}**%".format((countY/(len(df.Loan\_Status))\*100)))

print("Percentage of Rejected: **{:.2f}**%".format((countN/(len(df.Loan\_Status))\*100)))

print("Missing values percentage: **{:.2f}**%".format((countNull/(len(df.Loan\_Status))\*100)))

Percentage of Approved: 68.73%

Percentage of Rejected: 31.27%

Missing values percentage: 0.00%

👉 The number of approved loans is higher compared to rejected loans . It also can be seen, there is no missing values in this column.

### 4.1.9 Loan Amount Term

In [26]:

df.Loan\_Amount\_Term.value\_counts(dropna=False)

Out[26]:

360.0 512

180.0 44

480.0 15

NaN 14

300.0 13

240.0 4

84.0 4

120.0 3

60.0 2

36.0 2

12.0 1

Name: Loan\_Amount\_Term, dtype: int64

In [27]:

sns.countplot(x="Loan\_Amount\_Term",data=df,palette="rocket")

plt.show()

In [28]:

count12=len(df[df.Loan\_Amount\_Term==12.0])

count36=len(df[df.Loan\_Amount\_Term==36.0])

count60=len(df[df.Loan\_Amount\_Term==60.0])

count84=len(df[df.Loan\_Amount\_Term==84.0])

count120=len(df[df.Loan\_Amount\_Term==120.0])

count180=len(df[df.Loan\_Amount\_Term==180.0])

count240=len(df[df.Loan\_Amount\_Term==240.0])

count300=len(df[df.Loan\_Amount\_Term==300.0])

count360=len(df[df.Loan\_Amount\_Term==360.0])

count480=len(df[df.Loan\_Amount\_Term==480.0])

countNull=len(df[df.Loan\_Amount\_Term.isnull()])

print("Percentage of 12: **{:.2f}**%".format((count12/(len(df.Loan\_Amount\_Term))\*100)))

print("Percentage of 36: **{:.2f}**%".format((count36/(len(df.Loan\_Amount\_Term))\*100)))

print("Percentage of 60: **{:.2f}**%".format((count60/(len(df.Loan\_Amount\_Term))\*100)))

print("Percentage of 84: **{:.2f}**%".format((count84/(len(df.Loan\_Amount\_Term))\*100)))

print("Percentage of 120: **{:.2f}**%".format((count120/(len(df.Loan\_Amount\_Term))\*100)))

print("Percentage of 180: **{:.2f}**%".format((count180/(len(df.Loan\_Amount\_Term))\*100)))

print("Percentage of 240: **{:.2f}**%".format((count240/(len(df.Loan\_Amount\_Term))\*100)))

print("Percentage of 300: **{:.2f}**%".format((count300/(len(df.Loan\_Amount\_Term))\*100)))

print("Percentage of 360: **{:.2f}**%".format((count360/(len(df.Loan\_Amount\_Term))\*100)))

print("Percentage of 480: **{:.2f}**%".format((count480/(len(df.Loan\_Amount\_Term))\*100)))

print("Missing values percentage: **{:.2f}**%".format((countNull/(len(df.Loan\_Amount\_Term))\*100)))

Percentage of 12: 0.16%

Percentage of 36: 0.33%

Percentage of 60: 0.33%

Percentage of 84: 0.65%

Percentage of 120: 0.49%

Percentage of 180: 7.17%

Percentage of 240: 0.65%

Percentage of 300: 2.12%

Percentage of 360: 83.39%

Percentage of 480: 2.44%

Missing values percentage: 2.28%

👉As can be seen from the results, **the 360 days loan duration is the most popular** compared to others.

## **4.2 Numerical Variable**

👉The second variable that I will explore is categorical variable.

### 4.2.1 Describe Numerical Variable

👉This section will show mean, count, std, min, max and others using describe function.

In [29]:

df[['ApplicantIncome','CoapplicantIncome','LoanAmount']].describe()

Out[29]:

|  | ApplicantIncome | CoapplicantIncome | LoanAmount |
| --- | --- | --- | --- |
| count | 614.000000 | 614.000000 | 592.000000 |
| mean | 5403.459283 | 1621.245798 | 146.412162 |
| std | 6109.041673 | 2926.248369 | 85.587325 |
| min | 150.000000 | 0.000000 | 9.000000 |
| 25% | 2877.500000 | 0.000000 | 100.000000 |
| 50% | 3812.500000 | 1188.500000 | 128.000000 |
| 75% | 5795.000000 | 2297.250000 | 168.000000 |
| max | 81000.000000 | 41667.000000 | 700.000000 |

### 4.2.2 Distribution of Numerical Variable 📈

👉In this section, I will show the distribution of numerical variable using histogram and violin plot.

#### 4.2.2.1 Histogram Distribution 📉

In [30]:

sns.set(style="darkgrid")

fig,axs=plt.subplots(2,2,figsize=(10,8))

sns.histplot(data=df,x="ApplicantIncome",kde=True,ax=axs[0,0],color='green')

sns.histplot(data=df,x="CoapplicantIncome",kde=True,ax=axs[0,1],color='skyblue')

sns.histplot(data=df,x="LoanAmount",kde=True,ax=axs[1,0],color='orange');

#### 4.2.2.2 Violin Plot 🎻

In [31]:

sns.set(style="darkgrid")

fig,axs1=plt.subplots(2,2,figsize=(10,10))

sns.violinplot(data=df,y="ApplicantIncome",ax=axs1[0,0],color='green')

sns.violinplot(data=df,y="CoapplicantIncome",ax=axs1[0,1],color='skyblue')

sns.violinplot(data=df,y="LoanAmount",ax=axs1[1,0],color='orange');

* The distribution of **Applicant income, Co Applicant Income, and Loan Amount** are **positively skewed** and **it has outliers** (can be seen from both histogram and violin plot).
* The distribution of **Loan Amount Term** is **negativly skewed** and **it has outliers.**

## **4.3 Other Exploration**

👉This section will show additional exploration from each variables. The additional exploration are:

* Bivariate analysis (categorical w/ categorical, categroical w/ numerical, and numerical w/ numerical)
* Heatmap

### 4.3.1 Heatmap 🔥

In [32]:

plt.figure(figsize=(10,7))

sns.heatmap(df.corr(),annot=True,cmap='inferno');

👉There is positive correlation between Loan Amount and Applicant Income

### 4.3.2 Categorical - Categorical

In [33]:

pd.crosstab(df.Gender,df.Married).plot(kind="bar",stacked=True,figsize=(5,5),color=['#f64f59','#12c2e9'])

plt.title('Gender vs Married')

plt.xlabel('Gender')

plt.ylabel('Frequency')

plt.xticks(rotation=0)

plt.show()

👉Most male applicants are already married compared to female applicants. Also, the number of not married male applicants are higher compare to female applicants that had not married.

In [34]:

pd.crosstab(df.Self\_Employed,df.Credit\_History).plot(kind="bar",stacked=True,figsize=(5,5),color=['#544a7d','#ffd452'])

plt.title('Self Employed vs Credit History')

plt.xlabel('Self Employed')

plt.ylabel('Frequency')

plt.legend(["Bad Credit","Good Credit"])

plt.xticks(rotation=0)

plt.show()

👉Most not self employed applicants have good credit compared to self employed applicants.

In [35]:

pd.crosstab(df.Property\_Area,df.Loan\_Status).plot(kind="bar",stacked=True,figsize=(5,5),color=['#333333','#dd1818'])

plt.title('Property Area vs Loan Status')

plt.xlabel('Property Area')

plt.ylabel('Frequency')

plt.xticks(rotation=0)

plt.show()

👉 Most of loan that got accepted has property in Semiurban compared to Urban and Rural.

### 4.3.3 Categorical 📊- Numerical 📈

In [36]:

sns.boxplot(x="Loan\_Status",y="ApplicantIncome",data=df,palette="mako");

👉It can be seen that there are lots of outliers in Applicant Income, and the distribution also positively skewed

In [37]:

sns.boxplot(x="CoapplicantIncome",y="Loan\_Status",data=df,palette="rocket");

👉It's clear that Co Applicant Income has a number of outliers, and the distribution is also positively skewed.

In [38]:

sns.boxplot(x="Loan\_Status",y="LoanAmount",data=df,palette="YlOrBr");

👉As can be seen, Co Applicant Income has a high number of outliers, and the distribution is also positively skewed.

### 4.3.4 Numerical 📈 - Numerical 📈

In [39]:

df.plot(x='ApplicantIncome',y='CoapplicantIncome',style='o')

plt.title('Applicant Income - Co Applicant Income')

plt.xlabel('ApplicantIncome')

plt.ylabel('CoapplicantIncome')

plt.show()

print('Pearson correlation:',df['ApplicantIncome'].corr(df['CoapplicantIncome']))

print('T Test and P value: **\n**',stats.ttest\_ind(df['ApplicantIncome'],df['CoapplicantIncome']))

Pearson correlation: -0.11660458122889966

T Test and P value:

Ttest\_indResult(statistic=13.835753259915661, pvalue=1.4609839484240346e-40)

* There is **negative correlation** between Applicant income and Co Applicant Income.
* The correlation coefficient is **significant** at the 95 per cent confidence interval, as it has a **p-value of 1.46**

## **4.4 Null Values 🚫**

In [40]:

df.isnull().sum()

Out[40]:

Loan\_ID 0

Gender 13

Married 3

Dependents 15

Education 0

Self\_Employed 32

ApplicantIncome 0

CoapplicantIncome 0

LoanAmount 22

Loan\_Amount\_Term 14

Credit\_History 50

Property\_Area 0

Loan\_Status 0

dtype: int64

In [41]:

plt.figure(figsize=(24,5))

axz=plt.subplot(1,2,2)

mso.bar(df,ax=axz,fontsize=12);

👉Previously, the null values has been explored for Categorical Variables. In this section, the null values has been explored **for all variables** in the dataset.

# **5. Data Preprocessing**

## **5.1 Drop Unecessary Variables**

👉 Unecessary variables will be dropped in this section.

In [42]:

df=df.drop(['Loan\_ID'],axis=1)

## **5.2 Data Imputation**

👉 Imputation is a technique for substituting an estimated value for missing values in a dataset. In this section, the imputation will be performed for variables that have missing values.

### 5.2.1 Categorical Variables

👉In this section, the imputation for categorical variables will be performed using **mode**.

In [43]:

df['Gender'].fillna(df['Gender'].mode()[0],inplace=True)

df['Married'].fillna(df['Married'].mode()[0],inplace=True)

df['Dependents'].fillna(df['Dependents'].mode()[0],inplace=True)

df['Self\_Employed'].fillna(df['Self\_Employed'].mode()[0],inplace=True)

df['Credit\_History'].fillna(df['Credit\_History'].mode()[0],inplace=True)

df['Loan\_Amount\_Term'].fillna(df['Loan\_Amount\_Term'].mode()[0],inplace=True)

### 5.2.2 Numerical Variables

👉The next section is imputation for numerical variables using **mean**.

In [44]:

df['LoanAmount'].fillna(df['LoanAmount'].mean(),inplace=True)

## **5.3 One-hot Encoding**

👉In this section, I will **transform categorical variables into a form that could be provided by ML algorithms to do a better prediction.**

In [45]:

df=pd.get\_dummies(df)

*# Drop columns*

df=df.drop(['Gender\_Female','Married\_No','Education\_Not Graduate',

'Self\_Employed\_No','Loan\_Status\_N'],axis=1)

*# Rename columns name*

new={'Gender\_Male':'Gender','Married\_Yes':'Married',

'Education\_Graduate':'Education','Self\_Employed\_Yes':'Self\_Employed',

'Loan\_Status\_Y':'Loan\_Status'}

df.rename(columns=new,inplace=True)

## **5.3 Remove Outliers & Infinite values**

👉Since there are outliers, **the outliers will be removed**.

In [46]:

Q1=df.quantile(0.25)

Q3=df.quantile(0.75)

IQR=Q3-Q1

df=df[~((df<(Q1-1.5\*IQR))|(df>(Q3+1.5\*IQR))).any(axis=1)]

## **5.4 Skewed Distribution Treatment**

👉In previous section, it already shown that **distribution for ApplicantIncome, CoapplicantIncome, and LoanAmount is positively skewed**.  
👉 I will use **square root transformation** to normalized the distribution.

In [47]:

*# Square Root Transformation*

df.ApplicantIncome=np.sqrt(df.ApplicantIncome)

df.CoapplicantIncome=np.sqrt(df.CoapplicantIncome)

df.LoanAmount=np.sqrt(df.LoanAmount)

In [48]:

sns.set(style="darkgrid")

fig,axs=plt.subplots(2,2,figsize=(10,8))

sns.histplot(data=df,x="ApplicantIncome",kde=True,ax=axs[0,0],color='green')

sns.histplot(data=df,x="CoapplicantIncome",kde=True,ax=axs[0,1],color='skyblue')

sns.histplot(data=df,x="LoanAmount",kde=True,ax=axs[1,0],color='orange');

👉As can be seen, the distribution after using log transformation are much better compared to original distribution.

## **5.5 Features Separating**

👉 Dependent features (Loan\_Status) will be seperated from independent features.

In [49]:

X=df.drop(["Loan\_Status"],axis=1)

y=df["Loan\_Status"]

## **5.6 SMOTE Technique**

👉In previous exploration, it can be seen that **the number between approved and rejected loan is imbalanced**. In this section, **oversampling technique will be used to avoid overfitting**,

In [50]:

X,y=SMOTE().fit\_resample(X,y)

In [51]:

sns.set\_theme(style="darkgrid")

sns.countplot(y=y,data=df,palette="coolwarm")

plt.ylabel('Loan Status')

plt.xlabel('Total')

plt.show()

👉As can be seen, the distrubtion of Loan status are now **balanced**.

## **5.7 Data Normalization**

👉In this section, data normalization will be performed **to normalize the range of independent variables or features of data**.

In [52]:

X=MinMaxScaler().fit\_transform(X)

## **5.8 Splitting Data Set**

👉The data set will be split into **80% train and 20% test**.

In [53]:

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=0)

# **6. Models**

## **6.1 Logistic Regression**

In [54]:

LRclassifier=LogisticRegression(solver='saga',max\_iter=500,random\_state=1)

LRclassifier.fit(X\_train,y\_train)

y\_pred=LRclassifier.predict(X\_test)

print(classification\_report(y\_test,y\_pred))

print(confusion\_matrix(y\_test,y\_pred))

from sklearn.metrics import accuracy\_score

LRAcc=accuracy\_score(y\_pred,y\_test)

print('LR accuracy: **{:.2f}**%'.format(LRAcc\*100))

precision recall f1-score support

0 0.83 0.87 0.85 23

1 0.86 0.82 0.84 22

accuracy 0.84 45

macro avg 0.85 0.84 0.84 45

weighted avg 0.84 0.84 0.84 45

[[20 3]

[ 4 18]]

**LR accuracy: 84.44%**

## **6.2 K-Nearest Neighbour (KNN)**

In [55]:

scoreListknn=[]

for i **in** range(1,21):

KNclassifier=KNeighborsClassifier(n\_neighbors=i)

KNclassifier.fit(X\_train,y\_train)

scoreListknn.append(KNclassifier.score(X\_test,y\_test))

plt.plot(range(1,21),scoreListknn)

plt.xticks(np.arange(1,21,1))

plt.xlabel("K value")

plt.ylabel("Score")

plt.show()

KNAcc=max(scoreListknn)

print("KNN best accuracy: **{:.2f}**%".format(KNAcc\*100))

**KNN best accuracy: 91.11%**

## **6.3 Support Vector Machine (SVM)**

In [56]:

SVCclassifier=SVC(kernel='rbf',max\_iter=500)

SVCclassifier.fit(X\_train,y\_train)

y\_pred=SVCclassifier.predict(X\_test)

print(classification\_report(y\_test,y\_pred))

print(confusion\_matrix(y\_test,y\_pred))

fromsklearn.metricsimportaccuracy\_score

SVCAcc=accuracy\_score(y\_pred,y\_test)

print('SVC accuracy: **{:.2f}**%'.format(SVCAcc\*100))

precision recall f1-score support

0 0.87 0.87 0.87 23

1 0.86 0.86 0.86 22

accuracy 0.87 45

macro avg 0.87 0.87 0.87 45

weighted avg 0.87 0.87 0.87 45

[[20 3]

[ 3 19]]

SVC accuracy: 86.67%

## **6.4 Naive Bayes**

### 6.4.1 Categorical NB

In [57]:

NBclassifier1=CategoricalNB()

NBclassifier1.fit(X\_train,y\_train)

y\_pred=NBclassifier1.predict(X\_test)

print(classification\_report(y\_test,y\_pred))

print(confusion\_matrix(y\_test,y\_pred))

fromsklearn.metricsimportaccuracy\_score

NBAcc1=accuracy\_score(y\_pred,y\_test)

print('Categorical Naive Bayes accuracy: **{:.2f}**%'.format(NBAcc1\*100))

precision recall f1-score support

0 0.81 0.74 0.77 23

1 0.75 0.82 0.78 22

accuracy 0.78 45

macro avg 0.78 0.78 0.78 45

weighted avg 0.78 0.78 0.78 45

[[17 6]

[ 4 18]]

**Categorical Naive Bayes accuracy: 77.78%**

### 6.4.2 Gaussian NB

In [58]:

NBclassifier2=GaussianNB()

NBclassifier2.fit(X\_train,y\_train)

y\_pred=NBclassifier2.predict(X\_test)

print(classification\_report(y\_test,y\_pred))

print(confusion\_matrix(y\_test,y\_pred))

fromsklearn.metricsimportaccuracy\_score

NBAcc2=accuracy\_score(y\_pred,y\_test)

print('Gaussian Naive Bayes accuracy: **{:.2f}**%'.format(NBAcc2\*100))

precision recall f1-score support

0 0.77 0.87 0.82 23

1 0.84 0.73 0.78 22

accuracy 0.80 45

macro avg 0.81 0.80 0.80 45

weighted avg 0.80 0.80 0.80 45

[[20 3]

[ 6 16]]

**Gaussian Naive Bayes accuracy: 80.00%**

## **6.5 Decision Tree**

In [59]:

scoreListDT=[]

fori**in**range(2,21):

DTclassifier=DecisionTreeClassifier(max\_leaf\_nodes=i)

DTclassifier.fit(X\_train,y\_train)

scoreListDT.append(DTclassifier.score(X\_test,y\_test))

plt.plot(range(2,21),scoreListDT)

plt.xticks(np.arange(2,21,1))

plt.xlabel("Leaf")

plt.ylabel("Score")

plt.show()

DTAcc=max(scoreListDT)

print("Decision Tree Accuracy: **{:.2f}**%".format(DTAcc\*100))

Decision Tree Accuracy: 86.67%

## **6.6 Random Forest**

In [60]:

scoreListRF=[]

fori**in**range(2,25):

RFclassifier=RandomForestClassifier(n\_estimators=1000,random\_state=1,max\_leaf\_nodes=i)

RFclassifier.fit(X\_train,y\_train)

scoreListRF.append(RFclassifier.score(X\_test,y\_test))

plt.plot(range(2,25),scoreListRF)

plt.xticks(np.arange(2,25,1))

plt.xlabel("RF Value")

plt.ylabel("Score")

plt.show()

RFAcc=max(scoreListRF)

print("Random Forest Accuracy: **{:.2f}**%".format(RFAcc\*100))

Random Forest Accuracy: 93.33%

## **6.7 Gradient Boosting**

In [61]:

paramsGB={'n\_estimators':[100,200,300,400,500],

'max\_depth':[1,2,3,4,5],

'subsample':[0.5,1],

'max\_leaf\_nodes':[2,5,10,20,30,40,50]}

In [62]:

GB=RandomizedSearchCV(GradientBoostingClassifier(),paramsGB,cv=20)

GB.fit(X\_train,y\_train)

Out[62]:

RandomizedSearchCV(cv=20, estimator=GradientBoostingClassifier(),

param\_distributions={'max\_depth': [1, 2, 3, 4, 5],

'max\_leaf\_nodes': [2, 5, 10, 20, 30, 40

50],

'n\_estimators': [100, 200, 300, 400,

500],

'subsample': [0.5, 1]})

In [63]:

print(GB.best\_estimator\_)

print(GB.best\_score\_)

print(GB.best\_params\_)

print(GB.best\_index\_)

GradientBoostingClassifier(max\_depth=4, max\_leaf\_nodes=50, n\_estimators=400,

subsample=1)

0.8270833333333332

{'subsample': 1, 'n\_estimators': 400, 'max\_leaf\_nodes': 50, 'max\_depth': 4}

9

In [64]:

GBclassifier=GradientBoostingClassifier(subsample=0.5,n\_estimators=400,max\_depth=4,max\_leaf\_nodes=10)

GBclassifier.fit(X\_train,y\_train)

y\_pred=GBclassifier.predict(X\_test)

print(classification\_report(y\_test,y\_pred))

print(confusion\_matrix(y\_test,y\_pred))

fromsklearn.metricsimportaccuracy\_score

GBAcc=accuracy\_score(y\_pred,y\_test)

print('Gradient Boosting accuracy: **{:.2f}**%'.format(GBAcc\*100))

precision recall f1-score support

0 0.77 0.87 0.82 23

1 0.84 0.73 0.78 22

accuracy 0.80 45

macro avg 0.81 0.80 0.80 45

weighted avg 0.80 0.80 0.80 45

[[20 3]

[ 6 16]]

**Gradient Boosting accuracy: 80.00%**

# **7. Model Comparison**

In [65]:

compare=pd.DataFrame({'Model':['Logistic Regression','K Neighbors',

'SVM','Categorical NB',

'Gaussian NB','Decision Tree',

'Random Forest','Gradient Boost'],

'Accuracy':[LRAcc\*100,KNAcc\*100,SVCAcc\*100,

NBAcc1\*100,NBAcc2\*100,DTAcc\*100,

RFAcc\*100,GBAcc\*100]})

compare.sort\_values(by='Accuracy',ascending=False)

**OUTPUT:**

|  | Model | Accuracy |
| --- | --- | --- |
| 6 | Random Forest | 93.333333 |
| 1 | K Neighbors | 91.111111 |
| 2 | SVM | 86.666667 |
| 5 | Decision Tree | 86.666667 |
| 0 | Logistic Regression | 84.444444 |
| 4 | Gaussian NB | 80.000000 |
| 7 | Gradient Boost | 80.000000 |
| 3 | Categorical NB | 77.777778 |

* In general, it can be seen that **all models can achieve up to 70% accuracy**.  
  The highest accuracy is **93%%**.