**TASK-3**

**SALES PREDICTION USING PYTHON**

* **Sales prediction involves forecasting the amount of a product that**

**customers will purchase, taking into account various factors such as**

**advertising expenditure, target audience segmentation, and**

**advertising platform selection.**

* **In businesses that offer products or services, the role of a Data**

**Scientist is crucial for predicting future sales. They utilize machine**

**learning techniques in Python to analyze and interpret data, allowing**

**them to make informed decisions regarding advertising costs. By**

**leveraging these predictions, businesses can optimize their**

**advertising strategies and maximize sales potential. Let's embark on the journey of sales prediction using machine learning in Python.**

Sales prediction with Python enables businesses to forecast future sales based on factors like advertising expenses, customer segmentation, and marketing platforms. Leveraging machine learning techniques, companies can analyze historical data, build predictive models, and make data-driven decisions to optimize marketing strategies, allocate resources efficiently, and maximize revenue potential. By continuously updating the models with new data, businesses can adapt to market dynamics and achieve better performance in a competitive environment.

In [1]:

*# Import necessary libraries*

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import os

import statsmodels.formula.api as sm

from sklearn.linear\_model import LinearRegression, Ridge, Lasso, ElasticNet

from sklearn.metrics import mean\_squared\_error, r2\_score

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

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

import warnings

/opt/conda/lib/python3.10/site-packages/scipy/\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

warnings.warn(f"A NumPy version >={np\_minversion} and <{np\_maxversion}"

warnings.simplefilter(action='ignore', category=**FutureWarning**)

os.getcwd()

Out[2]:

'/kaggle/working'

In [3]:

*# Load dataset*

df = pd.read\_csv("/kaggle/input/advertisingcsv/Advertising.csv")

Exploratory Data Analysis

In [4]:

linkcode

*# View the first few rows*

*of the dataset*

df.head()

Out[4]:

|  | Unnamed: 0 | TV | Radio | Newspaper | Sales |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 2 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 3 | 17.2 | 45.9 | 69.3 | 9.3 |
| 3 | 4 | 151.5 | 41.3 | 58.5 | 18.5 |
| 4 | 5 | 180.8 | 10.8 | 58.4 | 12.9 |

*# Get the column names of the dataset*

df.columns

Out[5]:

Index(['Unnamed: 0', 'TV', 'Radio', 'Newspaper', 'Sales'], dtype='object')

In [6]:

*# To rename the column 'Unnamed: 0' to 'Index'*

df.rename(columns={'Unnamed: 0': 'Index'}, inplace=True)

In [7]:

linkcode

df

df

Out[7]:

|  | Index | TV | Radio | Newspaper | Sales |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 2 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 3 | 17.2 | 45.9 | 69.3 | 9.3 |
| 3 | 4 | 151.5 | 41.3 | 58.5 | 18.5 |
| 4 | 5 | 180.8 | 10.8 | 58.4 | 12.9 |
| ... | ... | ... | ... | ... | ... |
| 195 | 196 | 38.2 | 3.7 | 13.8 | 7.6 |
| 196 | 197 | 94.2 | 4.9 | 8.1 | 9.7 |
| 197 | 198 | 177.0 | 9.3 | 6.4 | 12.8 |
| 198 | 199 | 283.6 | 42.0 | 66.2 | 25.5 |
| 199 | 200 | 232.1 | 8.6 | 8.7 | 13.4 |

200 rows × 5 columns

*# Get the shape of the dataset (rows, columns)*

df.shape

Out[8]:

(200, 5)

In [9]:

*# Check information about the dataset, data types, and missing values*

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 200 entries, 0 to 199

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Index 200 non-null int64

1 TV 200 non-null float64

2 Radio 200 non-null float64

3 Newspaper 200 non-null float64

4 Sales 200 non-null float64

dtypes: float64(4), int64(1)

memory usage: 7.9 KB

In [10]:

*# Get statistical summary of the numerical columns*

df.describe().T

Out[10]:

|  | count | mean | std | min | 25% | 50% | 75% | max |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 200.0 | 100.5000 | 57.879185 | 1.0 | 50.750 | 100.50 | 150.250 | 200.0 |
| TV | 200.0 | 147.0425 | 85.854236 | 0.7 | 74.375 | 149.75 | 218.825 | 296.4 |
| Radio | 200.0 | 23.2640 | 14.846809 | 0.0 | 9.975 | 22.90 | 36.525 | 49.6 |
| Newspaper | 200.0 | 30.5540 | 21.778621 | 0.3 | 12.750 | 25.75 | 45.100 | 114.0 |
| Sales | 200.0 | 14.0225 | 5.217457 | 1.6 | 10.375 | 12.90 | 17.400 | 27.0 |

In [11]:

*# Check for missing values in the dataset*

df.isnull().values.any()

df.isnull().sum()

Index 0

TV 0

Radio 0

Newspaper 0

Sales 0

dtype: int64

Data Visualization

In [12]:

linkcode

*# Scatter plots to check the linearity assumption between each independent variable (TV, Radio, Newspaper) and the dependent variable (Sales)*

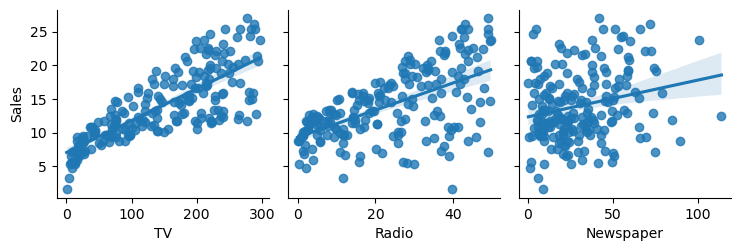
sns.pairplot(df, x\_vars=["TV", "Radio", "Newspaper"], y\_vars="Sales", kind="reg")

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

self.\_figure.tight\_layout(\*args, \*\*kwargs)

Out[12]:

<seaborn.axisgrid.PairGrid at 0x7e91db31e2c0>



*# Histograms to check the normality assumption of the dependent variable (Sales)*

df.hist(bins=20)

Out[13]:

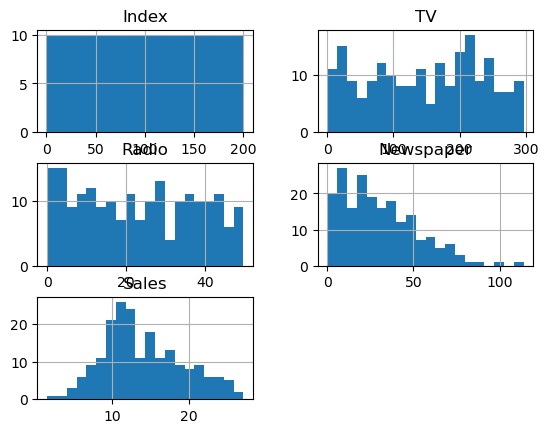
array([[<Axes: title={'center': 'Index'}>,

<Axes: title={'center': 'TV'}>],

[<Axes: title={'center': 'Radio'}>,

<Axes: title={'center': 'Newspaper'}>],

[<Axes: title={'center': 'Sales'}>, <Axes: >]], dtype=object)



*# Linear regression plots to visualize the relationship between each independent variable and the dependent variable*

sns.lmplot(x='TV', y='Sales', data=df)

sns.lmplot(x='Radio', y='Sales', data=df)

sns.lmplot(x='Newspaper',y= 'Sales', data=df)

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

self.\_figure.tight\_layout(\*args, \*\*kwargs)

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

self.\_figure.tight\_layout(\*args, \*\*kwargs)

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

self.\_figure.tight\_layout(\*args, \*\*kwargs)

Out[14]:

<seaborn.axisgrid.FacetGrid at 0x7e91d5f71c60>

SALES PREDICTION USING PYTHON

Sales prediction with Python enables businesses to forecast future sales based on factors like advertising expenses, customer segmentation, and marketing platforms. Leveraging machine learning techniques, companies can analyze historical data, build predictive models, and make data-driven decisions to optimize marketing strategies, allocate resources efficiently, and maximize revenue potential. By continuously updating the models with new data, businesses can adapt to market dynamics and achieve better performance in a competitive environment.

In [1]:

*# Import necessary libraries*

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import os

import statsmodels.formula.api as sm

from sklearn.linear\_model import LinearRegression, Ridge, Lasso, ElasticNet

from sklearn.metrics import mean\_squared\_error, r2\_score

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

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

import warnings

/opt/conda/lib/python3.10/site-packages/scipy/\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5

warnings.warn(f"A NumPy version >={np\_minversion} and <{np\_maxversion}"

In [2]:

warnings.simplefilter(action='ignore', category=**FutureWarning**)

os.getcwd()

Out[2]:

'/kaggle/working'

In [3]:

*# Load dataset*

df = pd.read\_csv("/kaggle/input/advertisingcsv/Advertising.csv")

Exploratory Data Analysis

In [4]:

*# View the first few rows of the dataset*

df.head()

Out[4]:

|  | Unnamed: 0 | TV | Radio | Newspaper | Sales |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 2 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 3 | 17.2 | 45.9 | 69.3 | 9.3 |
| 3 | 4 | 151.5 | 41.3 | 58.5 | 18.5 |
| 4 | 5 | 180.8 | 10.8 | 58.4 | 12.9 |

In [5]:

*# Get the column names of the dataset*

df.columns

Out[5]:

Index(['Unnamed: 0', 'TV', 'Radio', 'Newspaper', 'Sales'], dtype='object')

In [6]:

*# To rename the column 'Unnamed: 0' to 'Index'*

df.rename(columns={'Unnamed: 0': 'Index'}, inplace=True)

In [7]:

df

Out[7]:

|  | Index | TV | Radio | Newspaper | Sales |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 230.1 | 37.8 | 69.2 | 22.1 |
| 1 | 2 | 44.5 | 39.3 | 45.1 | 10.4 |
| 2 | 3 | 17.2 | 45.9 | 69.3 | 9.3 |
| 3 | 4 | 151.5 | 41.3 | 58.5 | 18.5 |
| 4 | 5 | 180.8 | 10.8 | 58.4 | 12.9 |
| ... | ... | ... | ... | ... | ... |
| 195 | 196 | 38.2 | 3.7 | 13.8 | 7.6 |
| 196 | 197 | 94.2 | 4.9 | 8.1 | 9.7 |
| 197 | 198 | 177.0 | 9.3 | 6.4 | 12.8 |
| 198 | 199 | 283.6 | 42.0 | 66.2 | 25.5 |
| 199 | 200 | 232.1 | 8.6 | 8.7 | 13.4 |

200 rows × 5 columns

In [8]:

*# Get the shape of the dataset (rows, columns)*

df.shape

Out[8]:

(200, 5)

In [9]:

*# Check information about the dataset, data types, and missing values*

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 200 entries, 0 to 199

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Index 200 non-null int64

1 TV 200 non-null float64

2 Radio 200 non-null float64

3 Newspaper 200 non-null float64

4 Sales 200 non-null float64

dtypes: float64(4), int64(1)

memory usage: 7.9 KB

In [10]:

*# Get statistical summary of the numerical columns*

df.describe().T

Out[10]:

|  | count | mean | std | min | 25% | 50% | 75% | max |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 200.0 | 100.5000 | 57.879185 | 1.0 | 50.750 | 100.50 | 150.250 | 200.0 |
| TV | 200.0 | 147.0425 | 85.854236 | 0.7 | 74.375 | 149.75 | 218.825 | 296.4 |
| Radio | 200.0 | 23.2640 | 14.846809 | 0.0 | 9.975 | 22.90 | 36.525 | 49.6 |
| Newspaper | 200.0 | 30.5540 | 21.778621 | 0.3 | 12.750 | 25.75 | 45.100 | 114.0 |
| Sales | 200.0 | 14.0225 | 5.217457 | 1.6 | 10.375 | 12.90 | 17.400 | 27.0 |

In [11]:

*# Check for missing values in the dataset*

df.isnull().values.any()

df.isnull().sum()

Out[11]:

Index 0

TV 0

Radio 0

Newspaper 0

Sales 0

dtype: int64

Data Visualization

In [12]:

*# Scatter plots to check the linearity assumption between each independent variable (TV, Radio, Newspaper) and the dependent variable (Sales)*

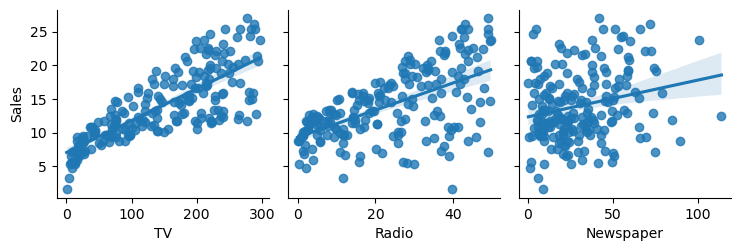
sns.pairplot(df, x\_vars=["TV", "Radio", "Newspaper"], y\_vars="Sales", kind="reg")

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

self.\_figure.tight\_layout(\*args, \*\*kwargs)

Out[12]:

<seaborn.axisgrid.PairGrid at 0x7e91db31e2c0>



In [13]:

*# Histograms to check the normality assumption of the dependent variable (Sales)*

df.hist(bins=20)

Out[13]:

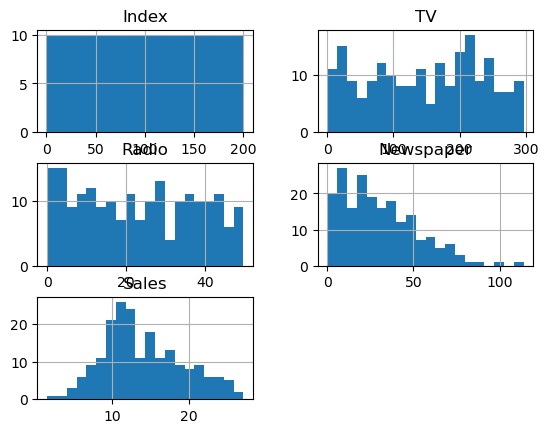
array([[<Axes: title={'center': 'Index'}>,

<Axes: title={'center': 'TV'}>],

[<Axes: title={'center': 'Radio'}>,

<Axes: title={'center': 'Newspaper'}>],

[<Axes: title={'center': 'Sales'}>, <Axes: >]], dtype=object)



In [14]:

linkcode

*# Linear regression plots to visualize the relationship between each independent variable and the dependent variable*

sns.lmplot(x='TV', y='Sales', data=df)

sns.lmplot(x='Radio', y='Sales', data=df)

sns.lmplot(x='Newspaper',y= 'Sales', data=df)

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

self.\_figure.tight\_layout(\*args, \*\*kwargs)

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

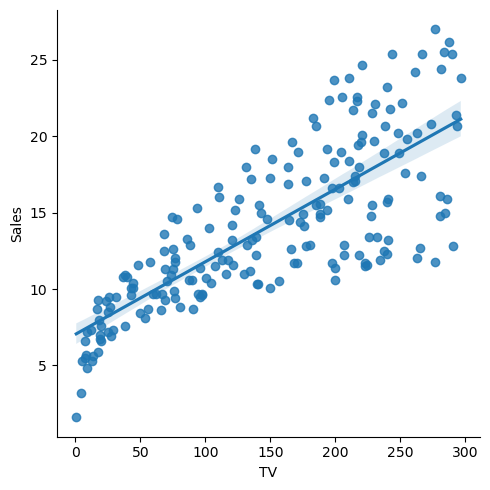
self.\_figure.tight\_layout(\*args, \*\*kwargs)

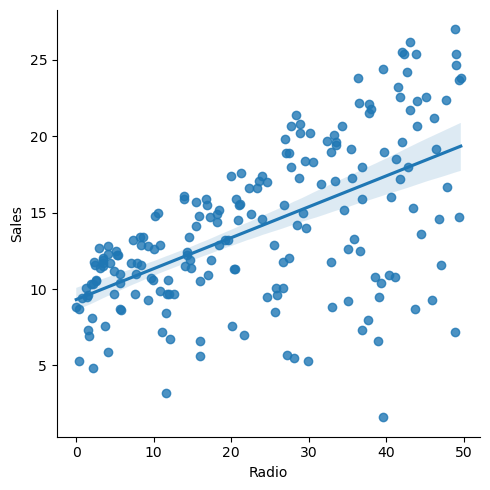
/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight

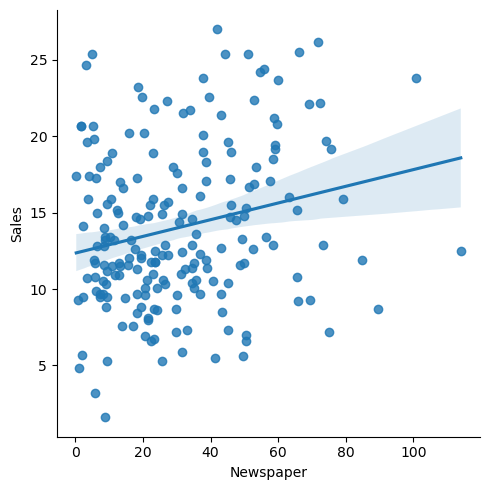
self.\_figure.tight\_layout(\*args, \*\*kwargs)

Out[14]:

<seaborn.axisgrid.FacetGrid at 0x7e91d5f71c60>







In [15]:

linkcode

*# Correlation Heatmap to check for multicollinearity among independent/dependent variable*

corrmat = df.corr()

f, ax = plt.subplots(figsize=(12, 9))

sns.heatmap(corrmat, vmin=0, vmax=1, square=True, cmap="YlGnBu", ax=ax)

plt.show()

A screenshot of a computer screen

Description automatically generated

*# Model Preparation*

X = df.drop('Sales', axis=1)

y = df[["Sales"]]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=46)

In [17]:

linkcode

*# Linear Regression Model*

lin\_model = sm.ols(formula="Sales ~ TV + Radio + Newspaper", data=df).fit()

In [18]:

*# Print the coefficients of the linear model*

print(lin\_model.params, "**\n**")

Intercept 2.938889

TV 0.045765

Radio 0.188530

Newspaper -0.001037

dtype: float64

In [19]:

linkcode

*# Print the summary of the linear regression model*

print(lin\_model.summary())

OLS Regression Results

==============================================================================

Dep. Variable: Sales R-squared: 0.897

Model: OLS Adj. R-squared: 0.896

Method: Least Squares F-statistic: 570.3

Date: Wed, 02 Aug 2023 Prob (F-statistic): 1.58e-96

Time: 12:08:46 Log-Likelihood: -386.18

No. Observations: 200 AIC: 780.4

Df Residuals: 196 BIC: 793.6

Df Model: 3

Covariance Type: nonrobust

==============================================================================

coef std err t P>|t| [0.025 0.975]

------------------------------------------------------------------------------

Intercept 2.9389 0.312 9.422 0.000 2.324 3.554

TV 0.0458 0.001 32.809 0.000 0.043 0.049

Radio 0.1885 0.009 21.893 0.000 0.172 0.206

Newspaper -0.0010 0.006 -0.177 0.860 -0.013 0.011

==============================================================================

Omnibus: 60.414 Durbin-Watson: 2.084

Prob(Omnibus): 0.000 Jarque-Bera (JB): 151.241

Skew: -1.327 Prob(JB): 1.44e-33

Kurtosis: 6.332 Cond. No. 454.

==============================================================================

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [20]:

linkcode

*# Evaluate the model*

results = []

names = []

In [21]:

*# Define a list of models to evaluate*

models = [('LinearRegression', LinearRegression())]

In [22]:

*# Loop through each model, fit it to the data, and calculate the RMSE*

for name, model **in** models:

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

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

result = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

results.append(result)

names.append(name)

msg = "**%s**: **%f**" % (name, result)

print(msg)

LinearRegression: 1.703648

In [23]:

linkcode

*# Make predictions on new data*

new\_data = pd.DataFrame({'TV': [100], 'Radio': [50], 'Newspaper': [25]})

predicted\_sales = lin\_model.predict(new\_data)

print("Predicted Sales:", predicted\_sales)

Predicted Sales: 0 16.915917

dtype: float64

*# Make predictions on new data*

new\_data = pd.DataFrame({'TV': [25], 'Radio': [63], 'Newspaper': [80]})

predicted\_sales = lin\_model.predict(new\_data)

print("Predicted Sales:", predicted\_sales)

Predicted Sales: 0 15.877397

dtype: float64

**CONCLUSION:**

**This is the overview of the task.**