Step - 1 : Business Problem Understanding

- **What is the relationship between each advertising channel (TV,Radio,Newspaper) and sales?**

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Step - 2 : Data Understanding

2.1. Load Data & Understand every variable

In [2]:

```
df = pd.read_csv("Advertising.csv")
df.head()
```

Out[2]:

	TV	radio	newspaper	sales
0	230100	37800	69200	22100
1	44500	39300	45100	10400
2	17200	45900	69300	9300
3	151500	41300	58500	18500
4	180800	10800	58400	12900

2.2. Dataset Understanding

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   TV         200 non-null    int64
 1   radio      200 non-null    int64
 2   newspaper  200 non-null    int64
 3   sales      200 non-null    int64
dtypes: int64(4)
memory usage: 6.4 KB
```

Step - 3 : Data Preprocessing

Exploratory Data Analysis

In [4]:

```
df.describe()
```

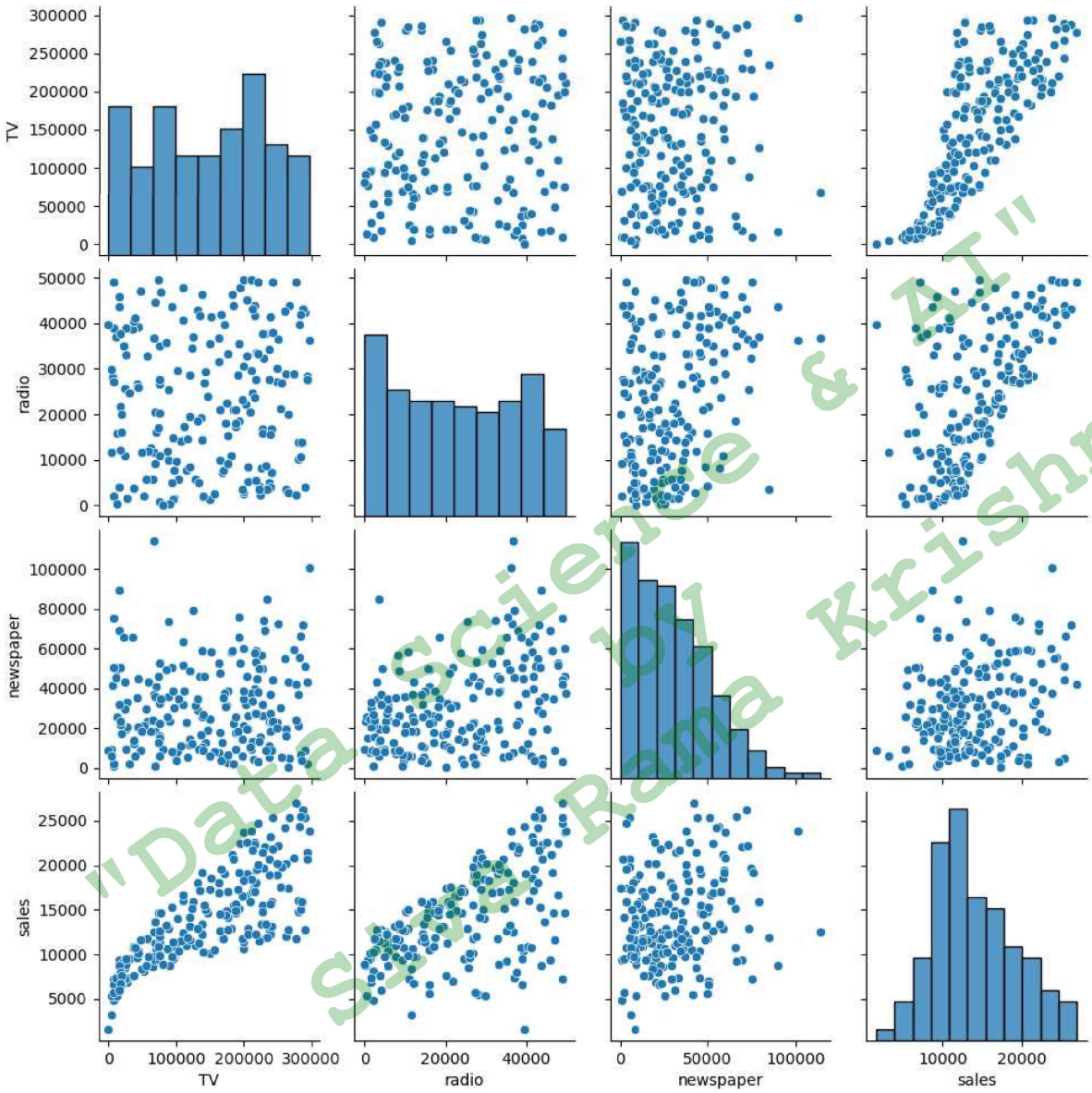
Out[4]:

	TV	radio	newspaper	sales
count	200.000000	200.000000	200.000000	200.000000
mean	147042.500000	23264.000000	30554.000000	14022.500000
std	85854.236315	14846.809176	21778.620839	5217.456566
min	700.000000	0.000000	300.000000	1600.000000
25%	74375.000000	9975.000000	12750.000000	10375.000000
50%	149750.000000	22900.000000	25750.000000	12900.000000
75%	218825.000000	36525.000000	45100.000000	17400.000000
max	296400.000000	49600.000000	114000.000000	27000.000000

"Data Science & AI" by Siva Rama Krishna

In [5]:

```
sns.pairplot(df)
plt.show()
```



"Data Science & AI" by Rama Krishna Sir

In [6]:

```
df.corr()
```

Out[6]:

	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

Data Cleaning

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
TV           0
radio        0
newspaper    0
sales        0
dtype: int64
```

In [8]:

```
#for this dataset, no data cleaning required
```

Data Wrangling

In [9]:

```
#for this dataset, no encoding required
```

X & y

In [10]:

```
X = df[["TV",'radio','newspaper']]
y = df['sales']
```

Train-Test-Split

In [11]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Modeling - Polynomial Regression

We will go from the equation in the form (shown here as if we only had one x feature):

$$\hat{y} = \beta_0 + \beta_1 x_1 + \epsilon$$

and create more features from the original x feature for some *d* degree of polynomial.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_1 x_1^2 + \ldots + \beta_d x_1^d + \epsilon$$

Then we can call the linear regression model on it, since in reality, we're just treating these new polynomial features x^2, x^3, ... x^d as new features. Obviously we need to be careful about choosing the correct value of *d* , the degree of the model. Our metric results on the test set will help us with this!

The other thing to note here is we have multiple X features, not just a single one as in the formula above, so in reality, the PolynomialFeatures will also take *interaction* terms into account for example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are [a, b, a^2, ab, b^2].

transform our original data set by adding polynomial features

In [12]:

```
from sklearn.preprocessing import PolynomialFeatures
polynomial_converter = PolynomialFeatures(degree=2,include_bias=False)
X_train = pd.DataFrame(polynomial_converter.fit_transform(X_train))
X_test = pd.DataFrame(polynomial_converter.transform(X_test))
```

In [13]:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

Out[13]:

```
LinearRegression()
```

In [14]:

```
model.intercept_
```

Out[14]:

```
4718.521644950224
```

In [15]:

```
model.coef_
```

Out[15]:

```
array([ 5.47817248e-02,  1.31121417e-02,  1.08639079e-02, -1.10740276e-07,
        1.08047633e-06, -1.07643423e-07,  3.01913511e-07,  1.75411787e-07,
        1.00748583e-07])
```

Predictions

"Data Science & AI" by Siva Rama Krishna

In [16]:

```
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)
```

Evaluation

In [17]:

```
model.score(X_train,y_train)  # Train R2
```

Out[17]:

```
0.9865054729019952
```

In [18]:

```
model.score(X_test,y_test)   # Test R2
```

Out[18]:

```
0.9808386009954223
```

Comparison with Linear Regression

Results on the Test Set (Note: Use the same Random Split to fairly compare!)

- Multiple Linear Regression:
  - R2: 0.9
- Polynomial 2-degree:
  - R2: 0.98

Everything in a single cell

In [19]:

```
#preprocessing
from sklearn.preprocessing import PolynomialFeatures
polynomial_converter = PolynomialFeatures(degree=2,include_bias=False)
X_poly = polynomial_converter.fit_transform(X)
X_poly = pd.DataFrame(X_poly)

#train_test spilt
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.3, random_state=42)

#modelling
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)

#prediction
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)

#Evaluation
print(model.score(X_train,y_train))
print(model.score(X_test,y_test))
```

```
0.9865054729019952
0.9808386009954223
```

In [20]:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model,X_poly,y,cv=5)
print(scores)
scores.mean()
```

```
[0.98795615 0.98937857 0.99129812 0.95889074 0.99374691]
```

Out[20]:

```
0.9842540981583106
```

HyperparameterTuning

Choosing the best polynomial degree for given dataset

Machine Science & AI" by Siva Rama Krishna

In [21]:

```
train_r2 =[]
test_r2=[]

for i in range(1,10):
    from sklearn.preprocessing import PolynomialFeatures
    polynomial_converter = PolynomialFeatures(degree=i,include_bias=False)
    X_poly = polynomial_converter.fit_transform(X)
    X_poly = pd.DataFrame(X_poly)

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.3, random_state=42)

    from sklearn.linear_model import LinearRegression
    model = LinearRegression()
    model.fit(X_train,y_train)

    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    train_r2.append(model.score(X_train,y_train))
    test_r2.append(model.score(X_test,y_test))
```

In [22]:

```
train_r2
```

Out[22]:

```
[0.9055159502227753,
 0.9865054729019952,
 0.9916220953950925,
 0.7611186831548564,
 0.9385531033488133,
 -17.365938230030938,
 0.9577475465901614,
 0.9667336830260811,
 0.9666938708637237]
```

In [23]:

```
test_r2
```

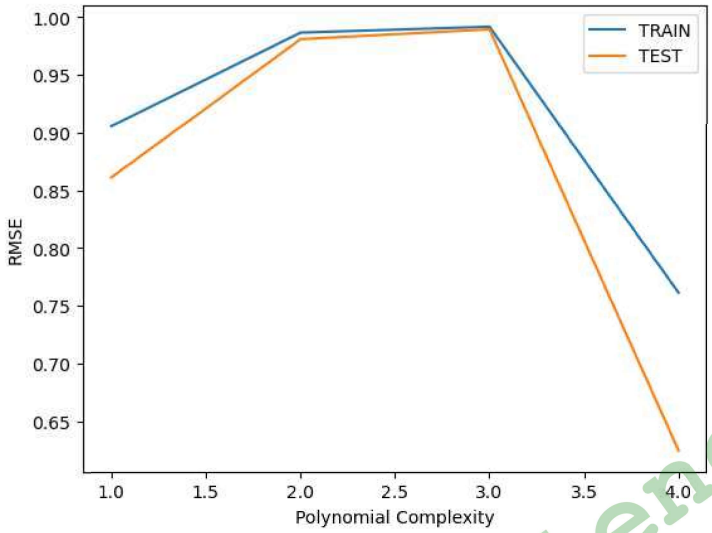
Out[23]:

```
[0.8609466508230369,
 0.9808386009954223,
 0.9893507535097348,
 0.6244141052697246,
 -2.2931014913654706,
 -14565.94881113971,
 -1608.2506131622106,
 -1458.0276745718845,
 -4811192.201467314]
```

Data Science & AI" by Siva Rama Krishna

In [24]:

```
plt.plot(range(1,5),train_r2[:4],label='TRAIN')
plt.plot(range(1,5),test_r2[:4],label='TEST')
plt.xlabel("Polynomial Complexity")
plt.ylabel("RMSE")
plt.legend()
plt.show()
```



Rebuilding the Model with best parameters

In [25]:

```
#preprocessing
final_poly_converter = PolynomialFeatures(degree=3,include_bias=False)
X_poly = final_poly_converter.fit_transform(X)

#train-test-split
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.3, random_state=42)

#build the final model
final_model = LinearRegression()
final_model.fit(X_train,y_train)

#prediction
train_pred = final_model.predict(X_train)
test_pred = final_model.predict(X_test)

# evalution
print("Train R2:",final_model.score(X_train,y_train)) # Train R2
print("Test R2:",final_model.score(X_test,y_test))   # Test R2
```

```
Train R2: 0.9916220953950925
Test R2: 0.9893507535097348
```

Prediction on New Data

Our next ad campaign will have a total spend of 149k on TV, 22k on Radio, and 12k on Newspaper Ads, how many units could we expect to sell as a result of this?

"Data Science & AI" by Siva Rama Krishna

In [26]:

```
input_data =[[149000,22000,12000]]

#predict
final_model.predict(input_data)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [26], line 4
      1 input_data =[[149000,22000,12000]]
      3 #predict
----> 4 final_model.predict(input_data)

File ~\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:362, in LinearModel.predict(self, X)
    348 def predict(self, X):
    349     """
    350     Predict using the linear model.
    351
   (...)
    360         Returns predicted values.
    361     """
--> 362     return self._decision_function(X)

File ~\anaconda3\lib\site-packages\sklearn\linear_model\_base.py:345, in LinearModel._decision_function(self, X)
    342 def _decision_function(self, X):
    343     check_is_fitted(self)
--> 345     X = self._validate_data(X, accept_sparse=["csr", "csc", "coo"], reset=False)
    346     return safe_sparse_dot(X, self.coef_.T, dense_output=True) + self.intercept_

File ~\anaconda3\lib\site-packages\sklearn\base.py:585, in BaseEstimator._validate_data(self, X, y, reset, validate_separat
ely, **check_params)
    582     out = X, y
    584 if not no_val_X and check_params.get("ensure_2d", True):
--> 585     self._check_n_features(X, reset=reset)
    587 return out

File ~\anaconda3\lib\site-packages\sklearn\base.py:400, in BaseEstimator._check_n_features(self, X, reset)
    397     return
    399 if n_features != self.n_features_in_:
--> 400     raise ValueError(
    401         f"X has {n_features} features, but {self.__class__.__name__} "
    402         f"is expecting {self.n_features_in_} features as input."
    403     )

ValueError: X has 3 features, but LinearRegression is expecting 19 features as input.
```

Recall that we will need to **convert** any incoming data to polynomial data, since that is what our model is trained on. We simply load up our saved converter object and only call **.transform()** on the new data, since we're not refitting to a new data set.

In [27]:

```
input_data =[[149000,22000,12000]]

#preprocesing
transformed_data = final_poly_converter.transform(input_data)

#predict
final_model.predict(transformed_data)
```

```
C:\Users\ADMIN\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but Polyn
omialFeatures was fitted with feature names
  warnings.warn(
```

Out[27]:

```
array([14577.25932079])
```

for deployment purpose we have to save the best model

There are now 2 things we need to save,

1. Polynomial feature creator
2. final Model with best degree value

Saving Model and Converter

In [28]:

```
from joblib import dump
```

"Data Science & AI" by Siva, Rama Krishna

In [29]:

```
dump(final_poly_converter,'poly_converter.joblib')
```

Out[29]:

```
['poly_converter.joblib']
```

In [30]:

```
dump(final_model, 'sales_poly_model.joblib')
```

Out[30]:

```
['sales_poly_model.joblib']
```

Load Model and Converter for prediction

In [31]:

```
from joblib import load
```

In [32]:

```
loaded_poly = load('poly_converter.joblib')
loaded_model = load('sales_poly_model.joblib')
```

In [33]:

```
#transform
campaign_poly = loaded_poly.transform([[149000,22000,12000]])

#predict
loaded_model.predict(campaign_poly)
```

```
C:\Users\ADMIN\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but Polyn
omialFeatures was fitted with feature names
  warnings.warn(
```

Out[33]:

```
array([14577.25932079])
```

"Data Science & AI" by Siva Rama Krishna