Advertising Sales Channel Prediction

1. Problem Definition.

When a company enters a market, the distribution strategy and channel it uses are keys to its success in the market, as well as market know-how and customer knowledge and understanding. Because an effective distribution strategy under efficient supply-chain management opens doors for attaining competitive advantage and strong brand equity in the market, it is a component of the marketing mix that cannot be ignored.

The distribution strategy and the channel design have to be right the first time. The case study of Sales channel includes the detailed study of TV, radio and newspaper channel.The predict the total sales generated from all the sales channel.

Hi, in this project I tried to create a prediction model for sales analysis. In this model, we need to feed the advertising budget the model will predict the possible sales. For designing the model the machine learning method I used is multiple regression model and the tool I used for coding is jupyter notebook.

The process for the model building will be completed in the following steps:

1. Importing all the required modules for the model

2. Extracting all the data from the dataset

3 Data cleaning and wrangling

4. Preparing training data for model

5. Preparing testing data for model

6. Creating, training and testing the model

7. Checking the accuracy of the model

8. Plotting the model graph to analyse

9. Saving the best model.

10. Conclusion.

About Project

Machine Learning Regression model is developed to predict sales based on budgesting spends on various platforms for predicting advertising sales. The features are TV, Radio and Newspaper marketing spend in thousands of dollars.

Packages used: Pandas, Numpy, Seaborn, Matplotlib, Scikit and Statsmodels.

Data fields

Features:

- TV: advertising dollars spent on TV for a single product in a given market (in thousands of dollars)
- Radio: advertising dollars spent on Radio
- Newspaper: advertising dollars spent on Newspaper

Target:

- Sales budget in thousands of dollars

2. Data Analysis

1. Importing all required modules.

Here the python modules I used are pandas, seaborn, sklearn, and matplotlib.

code

```
import pandas as pd

import seaborn as sns

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

from matplotlib import pyplot as plt
```

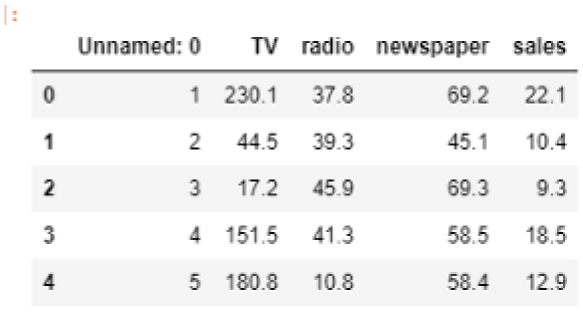
2. Extracting data from the dataset

The dataset was in the form of a CSV file, so I used the read_CSV file function from the pandas module. The picture of the dataset I just have given below, you can observe that it consists of four columns of TV advertising budget, Radio advertising budget, Newspaper Advertising budget and Sales records.

code

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

df.head()



```
   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
```

Understanding the Datasets

There are 4 columns in the dataset namely:

TV : It shows how much advertising sale of Television.

radio: It shows how much advertising sale of radio.

newspaper:It shows how much advertising sale of newspaper.

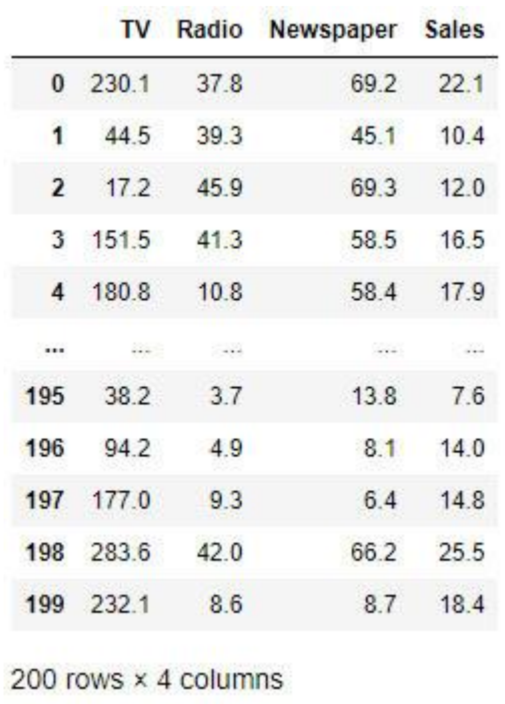
sales :Target column which predicting the sales of all advertising channel.

Instantly, you can see that the variable, Unnamed: 0, is essentially an index starting at 1 — so we're going to remove it.

Code

```
df = df.copy().drop(['Unnamed: 0'],axis=1)
```

After dropping we will get the following output.



	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	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	14.0
197	177.0	9.3	6.4	14.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	18.4

200 rows × 4 columns

3. Data cleaning and Wrangling

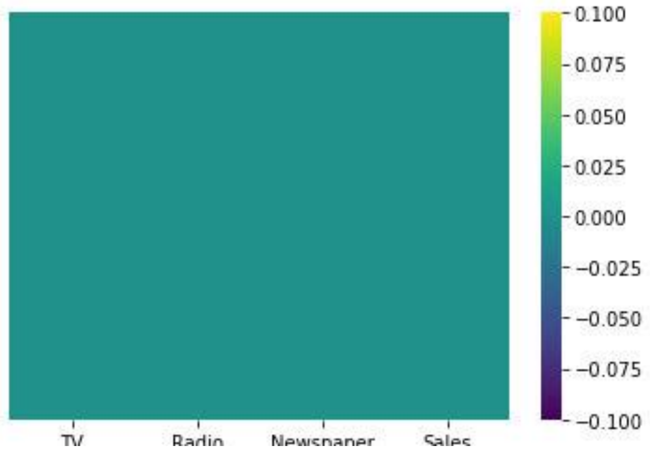
After extracting the data from the dataset now it's time to check the purity of the data. Try to find the missing values, null values which are hidden inside the dataset and remove that this will make the model more accurate.

To check the purity of the data set I use the heatmap function from the seaborn module. This will show the impurities hidden on which column.

code

```
sns.heatmap(read_data.isnull(),yticklabels=False,cmap='viridis')
```

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



Null values in columns

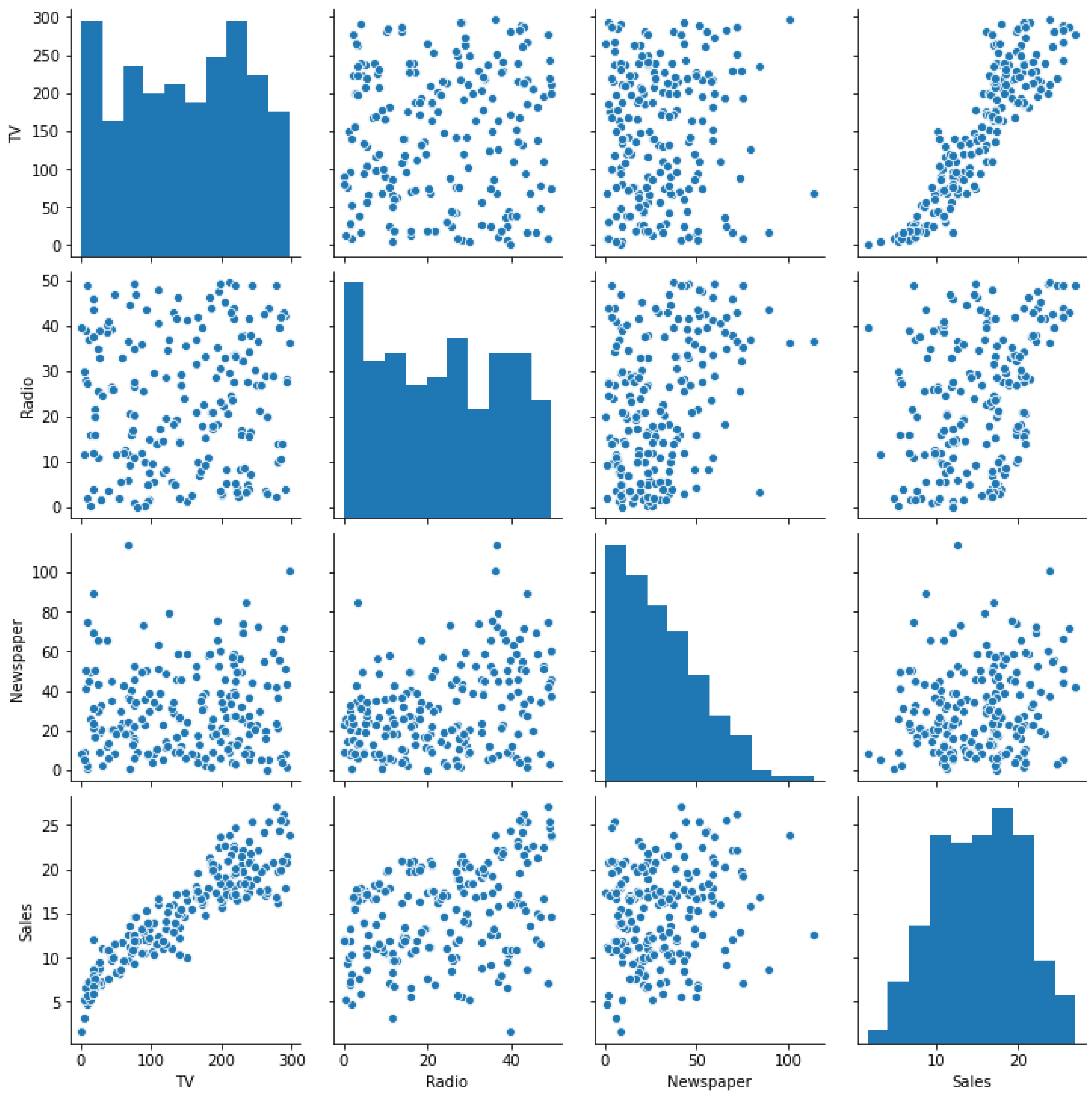
```
TV           0
radio        0
newspaper    0
sales        0
dtype: int64
```

Exploratory Data Analysis (EDA)

I looked at the distributions of the data and the value counts for the various numerical features. Below are a few highlights :

code

```
sns.pairplot( )
```



Observation:

From the above observation, it is clear that the data are not normalised and we should proceed with the data cleaning method.

Describe the data

Code

```
df.describe( )
```

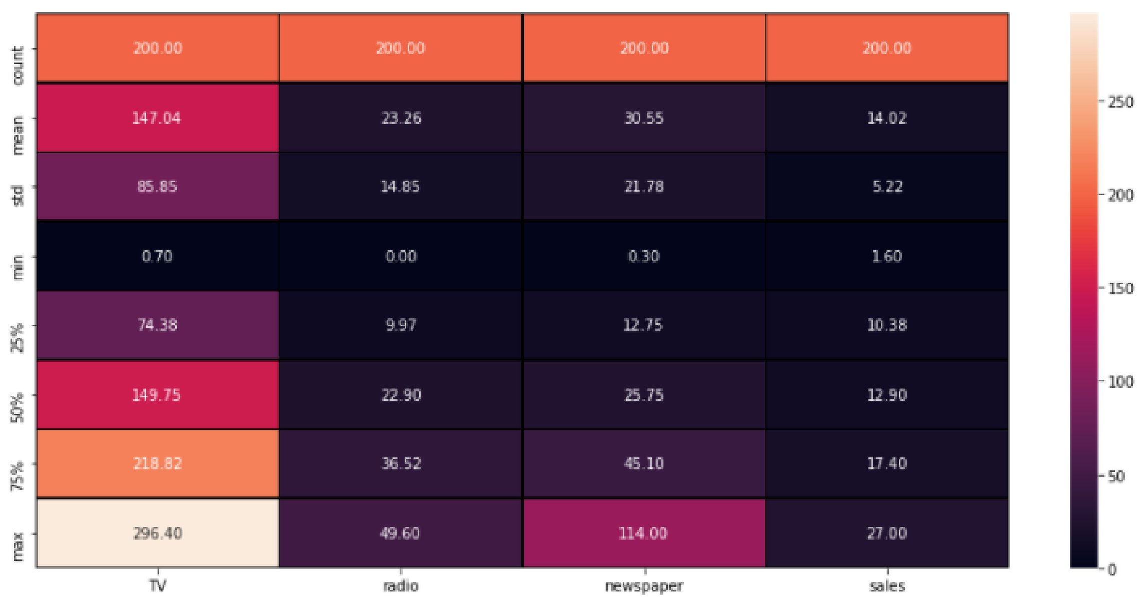
After this we will get the following output.

	TV	**radio**	**newspaper**	**sales**
count	200.000000	200.000000	200.000000	200.000000
mean	147.042500	23.264000	30.554000	14.022500
std	85.854236	14.846809	21.778621	5.217457
min	0.700000	0.000000	0.300000	1.600000
25%	74.375000	9.975000	12.750000	10.375000
50%	149.750000	22.900000	25.750000	12.900000
75%	218.825000	36.525000	45.100000	17.400000
max	296.400000	49.600000	114.000000	27.000000

To check describe data using heatmap fnction.

Code

```
plt.figure(figsize=(15,7))

sns.heatmap(df.describe(),annot=True,linewidth=0.5,linecolor='black',fmt='.2f')
```



From the above plotting we are determining mean, standard deviation, minimum and maximum value of each column. It helps us further in data cleaning.

TV:

Mean=147.042500

std=85.854236

max_value=296.4

min_value=0.7

radio

Mean=23.264000

std=14.846809

max_value=49.6

min_value=0

newspaper

Mean=30.554000

std=21.778621

max_value=114

min_value=0.3

sales

Mean=14.022500

std=5.217457

max_value=27

min_value=1.6

Checking the Correlation

Correlation analysis will determine the strength of a relationship between two continuous variables. We must compute the correlation coefficient to perform this analysis. Correlations take values between +1 and -1. A value close to 0 suggests a weak correlation between two variables.

Code

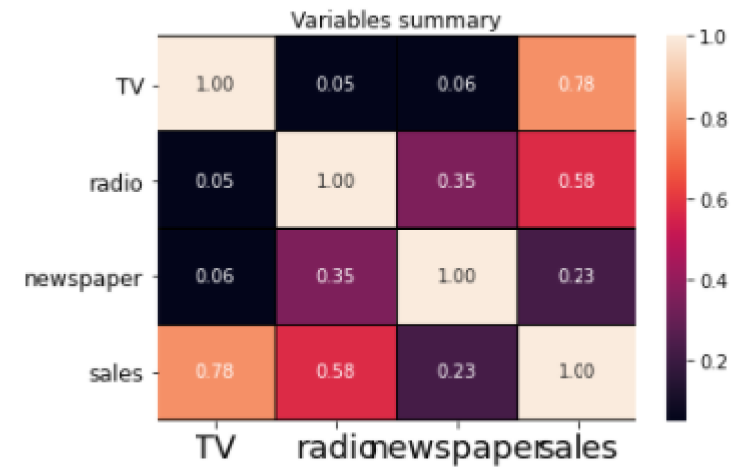
```
df.corr( )
```

	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

To check correlation using heatmap fnction

Code

```
plt.figure()

sns.heatmap(df.corr(),annot=True,linewidth=0.5,linecolor='black',fmt='.2f')

plt.xticks(fontsize=18)

plt.yticks(fontsize=12)

plt.title("Variables summary")

plt.show()
```



It is observed that all the columns are in good correlation with target column.

Data Transformation:

We need to transform those columns which has skewness.

Skewness

Code

```
df.skew( )
```

```
TV           -0.069853
radio         0.094175
newspaper     0.894720
sales         0.407571
dtype: float64
```

From the above observation the columns falls under high skewness:

newspaper = 0.894720

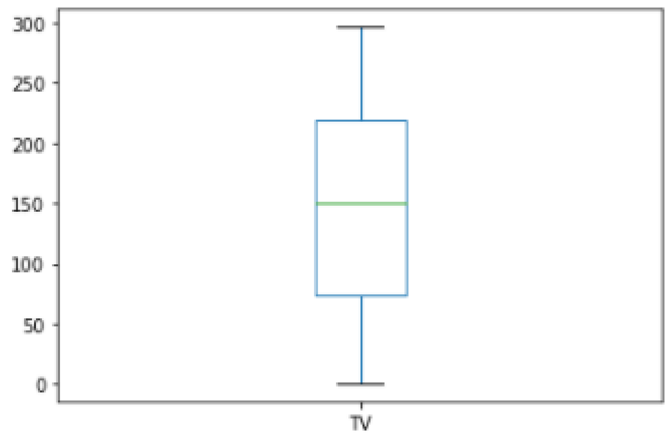
But we don’t drop it as it has good correlation with target column.

Now next step we need to check outliers.

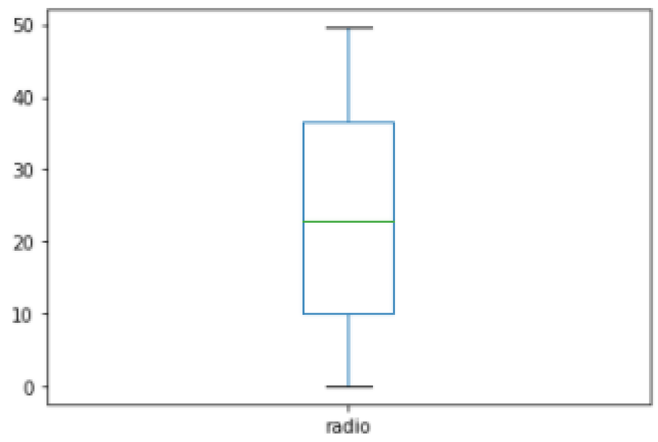
Outliers Check:

Code

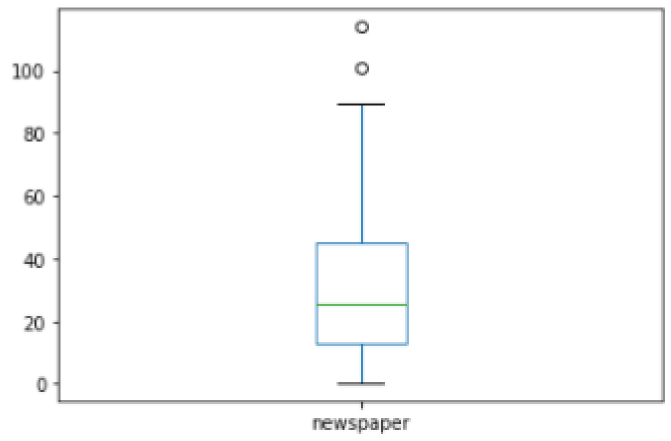
```
df['TV'].plot.box( )
```



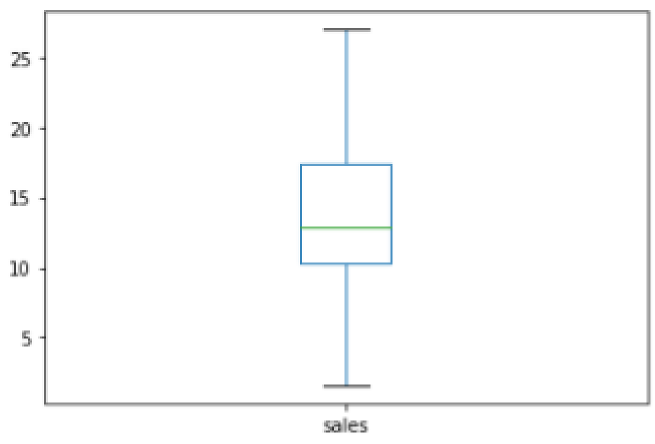
df['radio'].plot.box( )



df['newspaper'].plot.box( )



df['sales'].plot.box( )



from the above plot it is observed that some outliers are present before proceeding in training and testing data we need to remove the outliers.

Check the percentage of data falls under outliers:

Code:

```
from scipy.stats import zscore

import numpy as np

z=np.abs(zscore(df))

threshold=3

np.where(z>3)
```

Output:

```
(array([ 16, 101], dtype=int64), array([2, 2], dtype=int64))
```

So new dataframe is

	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

	TV	radio	newspaper	sales
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9
...	...	...	...	...
195	38.2	3.7	13.8	7.6
196	94.2	4.9	8.1	9.7
197	177.0	9.3	6.4	12.8
198	283.6	42.0	66.2	25.5
199	232.1	8.6	8.7	13.4

198 rows × 4 columns

3. EDA Concluding Remark.

All the features are in good correlation with target column. After finding skewness it is observed that the columns falls under high skewness is newspaper = 0.894720

But we don’t drop it as it has good correlation with target column. After plotting the boxplot it is observed that some outliers are present before proceeding in training and testing data we need to remove the outliers.

4. Building Machine Learning Models.

Generic Steps in model building using statsmodels

We first assign the feature variable, TV, Radio, newspaper to the variable X and the response variable, Sales, to the variable y

Model Building

Performing Simple Linear Regression

y=c+m1x1+m2x2+...+mnxn

- y is the response
- c is the intercept

- m1 is the coefficient for the first feature
- mn is the coefficient for the nth feature.

4. Preparing training and testing data for model

Code

```
from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

x_t=sc.fit_transform(x)

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

lr=LinearRegression()

from sklearn.metrics import r2_score

for selecting the best random state we need to check it using for loop.

max_scr=0

for i in range(0,1000):

    x_train,x_test,y_train,y_test=train_test_split(x_t,y,random_state=i,test_size=0.20)

    lr.fit(x_train,y_train)

    pred_train=lr.predict(x_train)

    pred_test=lr.predict(x_test)

    if round(r2_score(y_train,pred_train)*100,2)==round(r2_score(y_test,pred_test)*100,2):

        print("At random state",i," The model performs very well")

        print("At random_state:-",i)

        print("Training r2_score is:-", r2_score(y_train,pred_train)*100)
        print("Testing r2_score is:-", r2_score(y_test,pred_test)*100)
```

By doing this we get this output

```
At random state 112  The model performs very well
At random_state:- 112
Training r2_score is:- 90.17428654220274
Testing r2_score is:- 90.16708364952682
At random state 510  The model performs very well
At random_state:- 510
Training r2_score is:- 90.41100608643733
Testing r2_score is:- 90.41484328551749
```

hence we select random_state= 510 for training and testing.

6. Creating, training and testing the model

Train-Test Split

You now need to split our variable into training and testing sets. You'll perform this by importing train_test_split from the sklearn.model_selection library. It is usually a good practice to keep 70% of the data in your train dataset and the rest 30% in your test dataset

Code

```
x_train_b,x_test_b,y_train_b,y_test_b=train_test_split(x_t,y,random_state=510,test_size=0.2
0)
```

Regularization

Regularization is the most used technique to penalize complex models in machine learning, it is deployed for reducing over fitting (or, contracting generalization errors) by putting network weights small. Also, it enhances the performance of models for new inputs.

L1 and L2 regularization.

The main intuitive difference between the L1 and L2 regularization is that L1 regularization tries to estimate the median of the data while the L2 regularization tries to estimate the mean of the data to avoid overfitting. That value will also be the median of the data distribution mathematically.

1. Lasso:

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). ... The acronym “LASSO” stands for Least Absolute Shrinkage and Selection Operator.

Hyperparameter tuning

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process.

Grid search

Grid search is arguably the most basic hyperparameter tuning method. With this technique, we simply build a model for each possible combination of all of the hyperparameter values provided, evaluating each model, and selecting the architecture which produces the best results.

Code

```
from sklearn.linear_model import Lasso

parameters = {'alpha':[.0001, .001, .01, .1, 1, 10],'random_state':list(range(0,10))}

ls = Lasso()

clf = GridSearchCV(ls,parameters)

clf.fit(x_train,y_train)

print(clf.best_params_)
```

Output :-

```
{'alpha': 0.1, 'random_state': 0}
```

Cross-validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation.

Code

```
ls = Lasso(alpha=0.1,random_state=0)

ls.fit(x_train_b,y_train_b)

ls.score(x_train_b,y_train_b)

pred_ls = ls.predict(x_test_b)


lss = r2_score(y_test_b,pred_ls)
```

```
for j in range(2,10):
    lsscore = cross_val_score(ls,x_t,y,cv=j)
    lsc = lsscore.mean()
    print("At cv:-",j)
    print("Cross validation score is:-",lsc*100 )
    print("R2_score is :-",lss*100)
    print("\n")
```

Output:-

```
At cv:- 2
Cross validation score is:- 90.14616379006017
R2_score is :- 90.73427448807215


At cv:- 3
Cross validation score is:- 89.72351383573026
R2_score is :- 90.73427448807215


At cv:- 4
Cross validation score is:- 89.80785666814384
R2_score is :- 90.73427448807215


At cv:- 5
Cross validation score is:- 89.57805874289184
R2_score is :- 90.73427448807215


At cv:- 6
Cross validation score is:- 89.81682352909685
R2_score is :- 90.73427448807215


At cv:- 7
Cross validation score is:- 89.46701250358076
R2_score is :- 90.73427448807215


At cv:- 8
Cross validation score is:- 89.81744733610321
R2_score is :- 90.73427448807215


At cv:- 9
Cross validation score is:- 89.36472532637083
```

```
R2_score is :- 90.73427448807215
hence we selected cv=2
```

Errors in Regression

Linear regression most often uses mean-square error (MSE) to calculate the error of the model. MSE is calculated by: measuring the distance of the observed y-values from the predicted y-values at each value of x; ... calculating the mean of each of the squared distances.

Code:-

```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(y_test_b,pred_ls))

print('Mean Squared Error:',mean_squared_error(y_test_b,pred_ls))

print('Root Mean Square Error:',np.sqrt(mean_squared_error(y_test_b,pred_ls)))
```

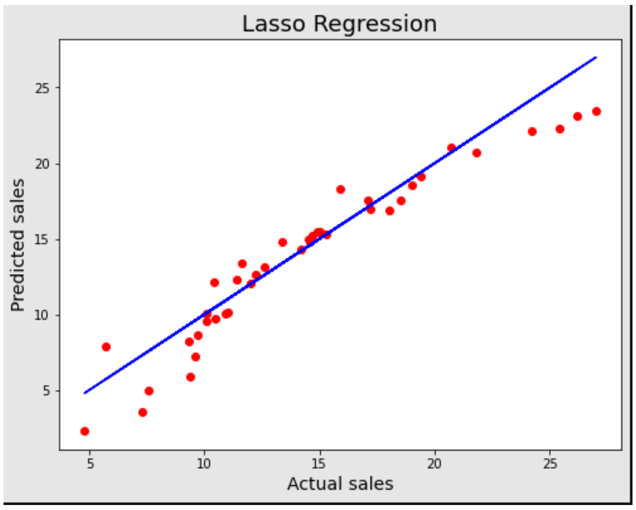
Output:-

```
Error:
Mean Absolute Error: 1.2554851904308837
Mean Squared Error: 2.7531342870846354
Root Mean Square Error: 1.6592571491738812
```

Visualizing the fit on the test set

Code:-

```
plt.figure(figsize=(8,6))
plt.scatter(x=y_test_b, y=pred_ls, color='r')
plt.plot(y_test_b,y_test_b, color='b')
plt.xlabel('Actual sales',fontsize=14)
plt.ylabel('Predicted sales',fontsize=14)
plt.title('Lasso Regression',fontsize=18)
plt.show()
```



The above results shows that the datapoints near the best fit line. Trying model testing with different algorithm

2. Ridge Regression

Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values to be far away from the actual values.

Hyperparameter tuning

Code:-

```
from sklearn.linear_model import Ridge

parameters = {'alpha':[.0001, .001, .01, .1,1], 'fit_intercept':[True,False],'normalize':[True,False],'copy_X':[True,False],'tol':[0.001,0.01,0.1],'random_state':[0,1,2,3,4,5,6,7,8,9]}

rd = Ridge()

clf = GridSearchCV(rd,parameters)

clf.fit(x_train_b,y_train_b)

print(clf.best_params_)
```

Output:-

```
{'alpha': 0.01, 'copy_X': True, 'fit_intercept': True, 'normalize': True, 'random_state': 0, 'tol': 0.001}
```

Cross-validation

```
rd = Ridge(alpha=0.01, copy_X= True, fit_intercept= True, normalize=True, random_state= 0, tol= 0.001)

rd.fit(x_train_b,y_train_b)

rd.score(x_train_b,y_train_b)

pred_rd = rd.predict(x_test_b)

rds = r2_score(y_test_b,pred_rd)

print('R2 Score:',rds*100)

rdscore = cross_val_score(rd,x_t,y,cv=3)

rdc = rdscore.mean()

print('Cross Val Score:',rdc*100)
```

Output:-

```
R2 Score: 90.53994674638164
Cross Val Score: 89.73959487390134
```

Error:-

Code-

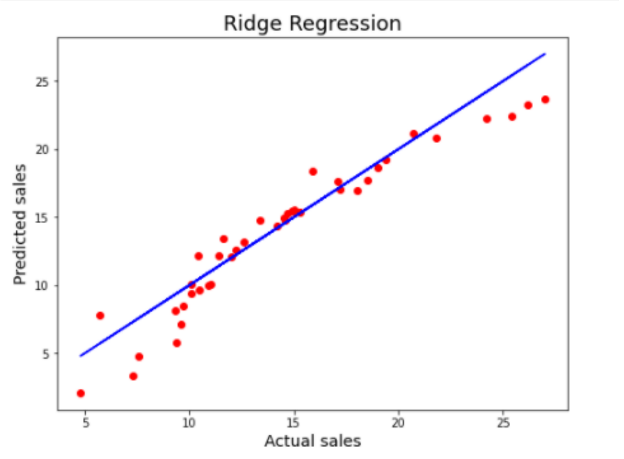
```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(y_test_b,pred_rd))

print('Mean Squared Error:',mean_squared_error(y_test_b,pred_rd))

print('Root Mean Square Error:',np.sqrt(mean_squared_error(y_test_b,pred_rd)))
```

Output:-

```
Error:
Mean Absolute Error: 1.2704347667414542
Mean Squared Error: 2.8108750833008727
Root Mean Square Error: 1.6765664565715468
```

Visualizing the fit on the test set
Code:-

```
plt.figure(figsize=(8,6))
plt.scatter(x=y_test_b, y=pred_rd, color='r')
plt.plot(y_test_b,y_test_b, color='b')
plt.xlabel('Actual sales',fontsize=14)
plt.ylabel('Predicted sales',fontsize=14)
plt.title('Lasso Regression',fontsize=18)
plt.show()
```



Ensemble Techniques:

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. To better understand this definition lets take a step back into ultimate goal of machine learning and model building.

For Decision Tree regressor

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes.

Code:

```
from sklearn.tree import DecisionTreeRegressor

parameters = {'criterion':['mse', 'friedman_mse', 'mae'], 'splitter':['best', 'random']}

dt =DecisionTreeRegressor()

clf = GridSearchCV(dt,parameters)

clf.fit(x_train_b,y_train_b)

print(clf.best_params_)
```

Output:

```
{'criterion': 'mse', 'splitter': 'random'}
```

Cross- validation

```
dt = DecisionTreeRegressor(criterion='friedman_mse', splitter='best')

dt.fit(x_train_b, y_train_b)

dt.score(x_train_b, y_train_b)

pred_decision = dt.predict(x_test_b)

dts = r2_score(y_test_b,pred_decision)

print('R2 Score:',dts*100)

dtscore = cross_val_score(dt,x_t,y,cv=3)

dtc = dtscore.mean()

print('Cross Val Score:',dtc*100)
```

Output

```
R2 Score: 94.96518370684984
Cross Val Score: 94.53060671561319
```

Error

```
print('Error:')

print('Mean Absolute Error:',mean_absolute_error(y_test_b,pred_decision))

print('Mean Squared Error:',mean_squared_error(y_test_b,pred_decision))

print('Root Mean Square Error:',np.sqrt(mean_squared_error(y_test_b,pred_decision)))
```

Output

```
Error:
Mean Absolute Error: 0.9199999999999999
Mean Squared Error: 1.4959999999999998
Root Mean Square Error: 1.2231107881136523
```

Random Forest regressor

A random forest regressor. A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The number of trees in the forest.

Code:

```
from sklearn.ensemble import RandomForestRegressor

parameters = {'criterion':['friedman_mse', 'mae'],'n_estimators':[100,200,300]}

rf = RandomForestRegressor()

clf = GridSearchCV(rf,parameters)

clf.fit(x_train_b,y_train_b)

print(clf.best_params_)
```

Output:

```
{'criterion': 'mae', 'n_estimators': 200}
```

Cross-Validation

```
rf = RandomForestRegressor(criterion='mae',n_estimators=200)

rf.fit(x_train_b, y_train_b)
rf.score(x_train_b, y_train_b)

pred_random = rf.predict(x_test_b)

rfs = r2_score(y_test_b,pred_random)

print('R2 Score:',rfs*100)

rfscore = cross_val_score(rf,x_t,y,cv=3)

rfc = rfscore.mean()

print('Cross Val Score:',rfc*100)
```

Outut:

```
R2 Score: 98.42155663242809
Cross Val Score: 97.47239588237608
```

GradientBoostingRegressor

Gradient boosting is a type of machine learning boosting. It relies on the intuition that the b est possible next model, when combined with previous models, minimizes the overall predicti on error. If a small change in the prediction for a case causes no change in error, then next tar get outcome of the case is zero.

Code:

```
from sklearn.datasets import make_regression
from sklearn.ensemble import GradientBoostingRegressor
parameters = {'loss' : ['ls', 'lad', 'huber', 'quantile'],'n_estimators':[50,100,200],'criterion':['fried
man_mse', 'mse']}
gbr=GradientBoostingRegressor()
clf = GridSearchCV(gbr,parameters)
clf.fit(x_train_b,y_train_b)
print(clf.best_params_)
```

Output:

```
{'criterion': 'mse', 'loss': 'ls', 'n_estimators': 200}
```

Cross-Validation

```
gbr= GradientBoostingRegressor(criterion='friedman_mse',loss='ls',n_estimators=200)
gbr.fit(x_train_b, y_train_b)
gbr.score(x_train_b, y_train_b)
pred_random = rf.predict(x_test_b)

gbrs= r2_score(y_test_b,pred_random)
print('R2 Score:',gbrs*100)

gbscore = cross_val_score(gbr,x_t,y,cv=3)
gbrc= gbscore.mean()
print('Cross Val Score:',gbrc*100)
```

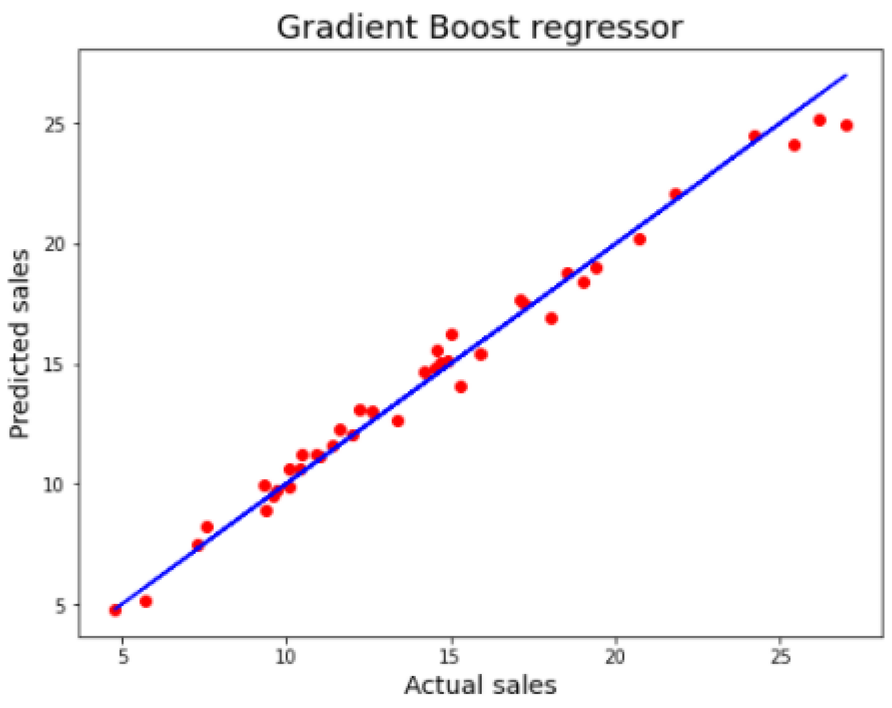
Output

```
R2 Score: 98.42155663242809
Cross Val Score: 97.70650071770685
```

Plotting the testing fit data

```
plt.figure(figsize=(8,6))
plt.scatter(x=y_test_b, y=pred_random, color='r')
plt.plot(y_test_b,y_test_b, color='b')
plt.xlabel('Actual sales',fontsize=14)
plt.ylabel('Predicted sales',fontsize=14)
plt.title('Gradient Boost regressor',fontsize=18)
plt.show()
```

Output:



The best model is Gradient Boost regressor. Since the difference between the percentage score of cross validation and r2_score is optimum.

Saving the best model.

code

```
import pickle
filename = 'in_advertising.pkl'
pickle.dump(gbr, open(filename, 'wb'))
```

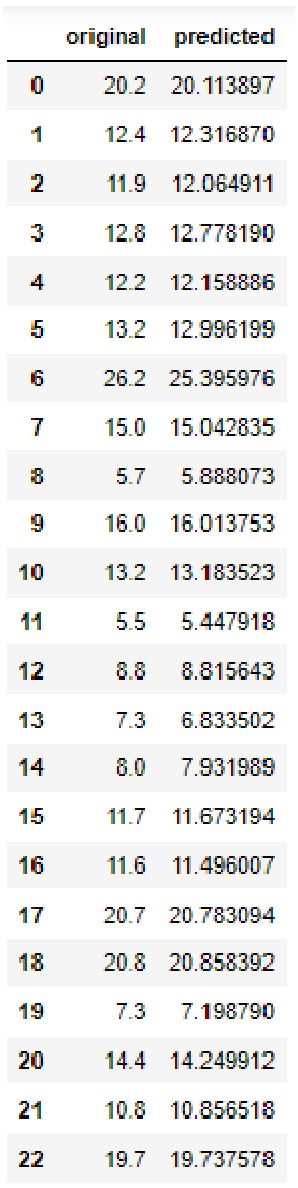
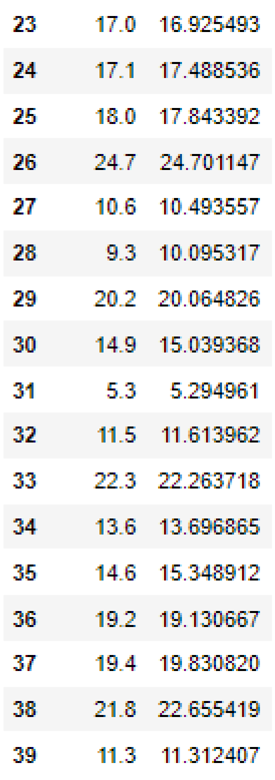
6. Concluding Remarks.

Finally we concluding the model by interpreting the test data again and by comparing the testing data and predicting data.

Code:

```
a=np.array(y_test)

predicted=np.array(gbr.predict(x_test))

df_con=pd.DataFrame({"original":a,"predicted":predicted}, index= range(len(a)))

df_con
```

Outut:

	original	predicted
0	20.2	20.113897
1	12.4	12.316870
2	11.9	12.064911
3	12.8	12.778190
4	12.2	12.158886
5	13.2	12.996199
6	26.2	25.395976
7	15.0	15.042835
8	5.7	5.888073
9	16.0	16.013753
10	13.2	13.183523
11	5.5	5.447918
12	8.8	8.815643
13	7.3	6.833502
14	8.0	7.931989
15	11.7	11.673194
16	11.6	11.496007
17	20.7	20.783094
18	20.8	20.858392
19	7.3	7.198790
20	14.4	14.249912
21	10.8	10.856518
22	19.7	19.737578
23	17.0	16.925493
24	17.1	17.488536
25	18.0	17.843392
26	24.7	24.701147
27	10.6	10.493557
28	9.3	10.095317
29	20.2	20.064826
30	14.9	15.039368
31	5.3	5.294961
32	11.5	11.613962
33	22.3	22.263718
34	13.6	13.696865
35	14.6	15.348912
36	19.2	19.130667
37	19.4	19.830820
38	21.8	22.655419
39	11.3	11.312407

From the above table the model is predicted the values with 96 percent accuracy