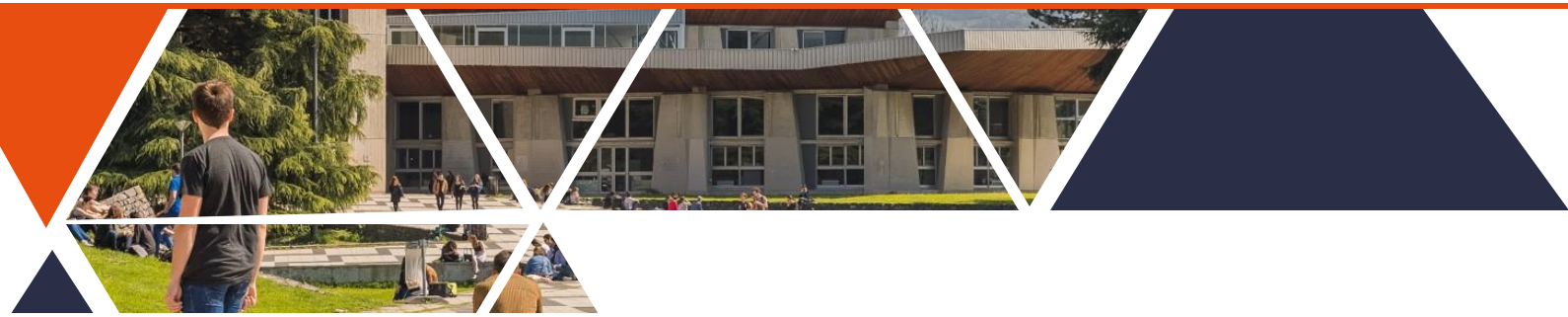
Python Programming
Data Frame- Linear Regression



Parcours Progis

Etudes, Medias, communication, Marketing

Bahareh Afshinpour.

21.11.2024

Seaborn.countplot()

-Seaborn is an amazing visualization library for statistical graphics plotting in Python.

-Seaborn.countplot() method is used to Show the **counts of observations** in each categorical bin using bars.

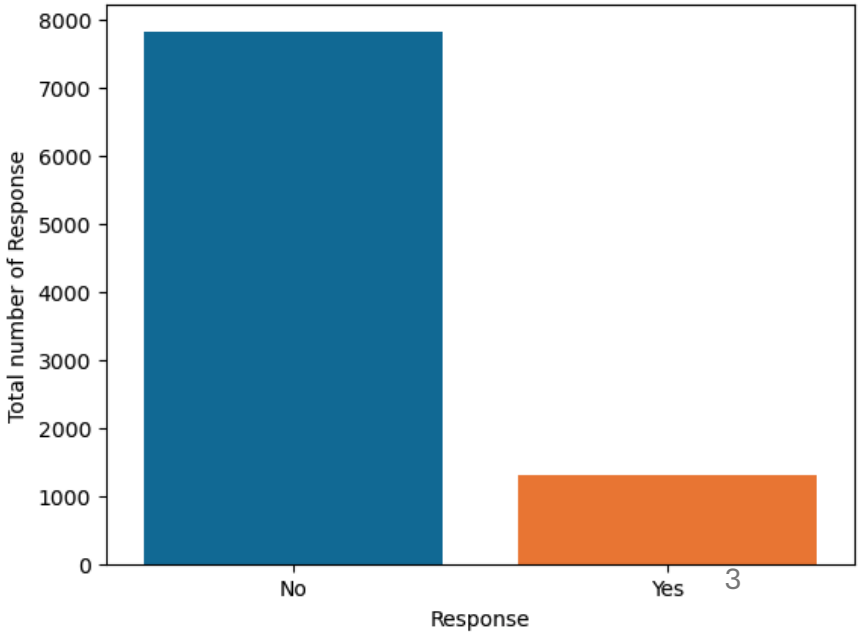
```
import seaborn as sns           # Provides a high level interface
ax = sns.countplot(x='Response',data = data)
plt.ylabel('Total number of Response')
annot_plot(ax, 0.08,1)
plt.show()
```

Groupby

The groupby() method allows you to group your data and execute functions on these groups.

```
Response=data.groupby('Response')['Customer'].count()
Response
```

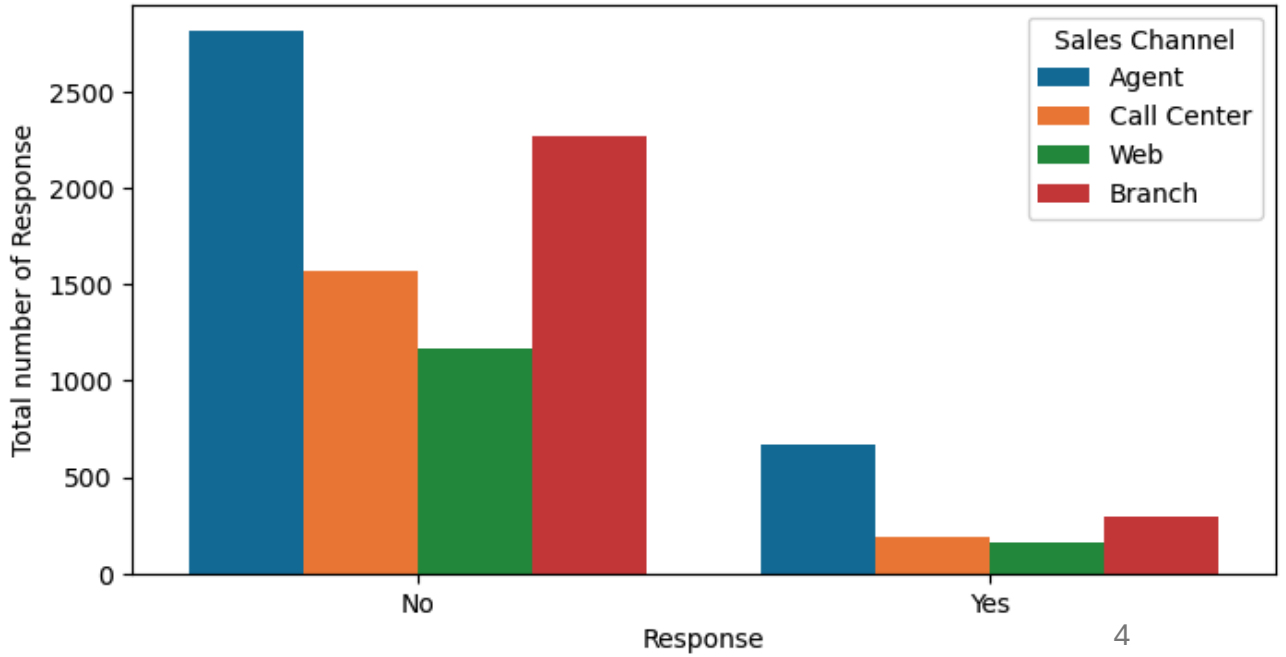
```
Response
No      7826
Yes     1308
Name: Customer, dtype: int64
```



The ratio of sales channel to response

```
plt.figure(figsize=(8,4))
ax = sns.countplot(x='Response',hue = 'Sales Channel' ,data = data)
plt.ylabel('Total number of Response')
annot_plot(ax, 0.08,1)
plt.show()
```

More than half of the engaged customers to respond are from « agent »

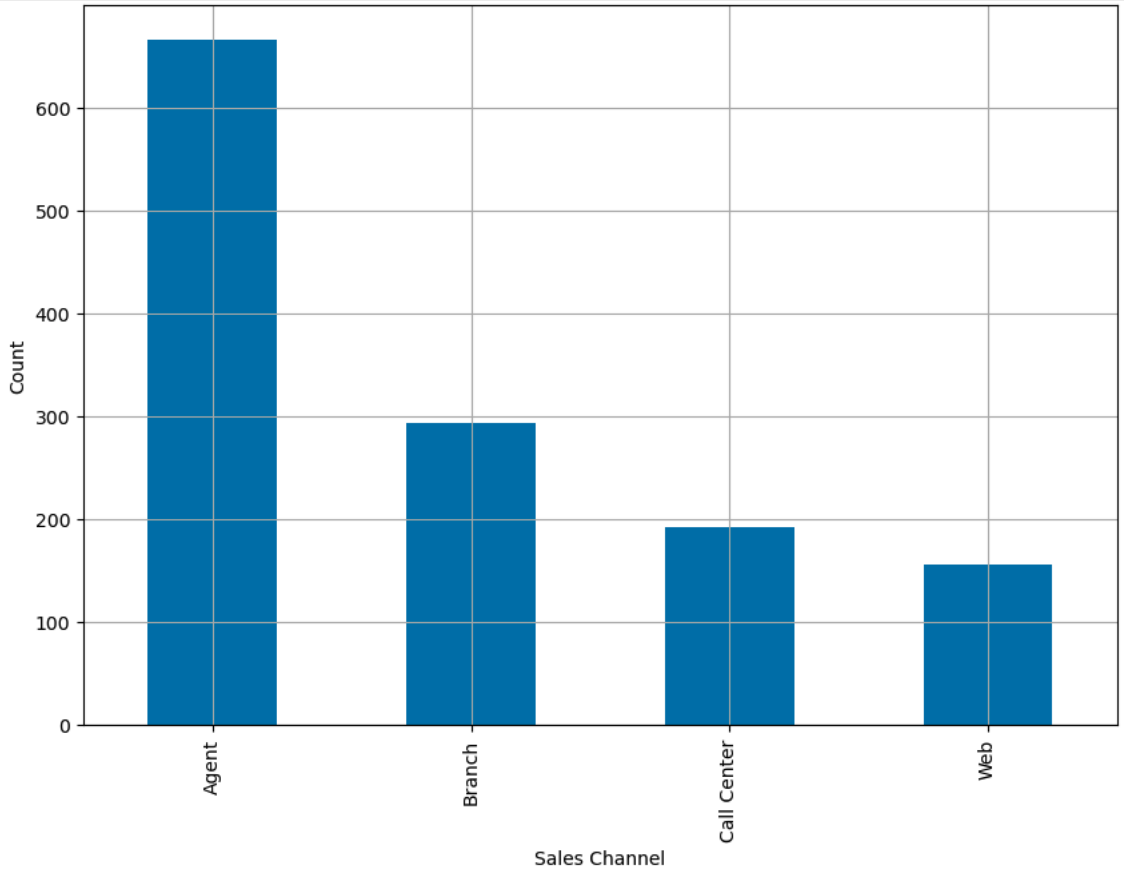


More about Sales Channel to response

```
channel_vs_mstatus=data[data['Response']=='Yes'].groupby(['Sales Channel'])['Customer'].count()
channel_vs_mstatus
```

```
Sales Channel
Agent          666
Branch         294
Call Center    192
Web            156
Name: Customer, dtype: int64
```

```
ax = (channel_vs_mstatus).plot(
kind='bar',
figsize=(10, 7),
grid=True
)
ax.set_ylabel('Count')
plt.show()
```



Select_dtypes (1)

- Here, the goal is to split the data into two parts, **numeric** and **categorical**.
- continous_var_data = data.select_dtypes(include=['int64','float'])

```
num=data.select_dtypes(include='number')
num.head()
```

	Customer Lifetime Value	Income	Monthly Premium Auto	Months Since Last Claim
0	2763.519279	56274	69	32
1	6979.535903	0	94	13
2	12887.431650	48767	108	18
3	7645.861827	0	106	18
4	2813.692575	43836	73	12

```
cate=data.select_dtypes(include='object')
cate.head()
```

	Customer	State	Response	Coverage	Education	Effective To Date	Empl
0	BU79786	Washington	No	Basic	Bachelor	2/24/11	
1	QZ44356	Arizona	No	Extended	Bachelor	1/31/11	

Select_dtypes (2)

num=data.select_dtypes(include='number')

num.head()

```
num=data.select_dtypes(include='number')
num.head()
```

	Customer Lifetime Value	Income	Monthly Premium Auto	Months Since Last Claim	Months Since Policy Inception
0	2763.519279	56274	69	32	5
1	6979.535903	0	94	13	42
2	12887.431650	48767	108	18	38
3	7645.861827	0	106	18	65
4	2813.692575	43836	73	12	44

Split the data into Train and Test datasets

- The train-test split is a technique for evaluating the performance of a machine learning algorithm.
- It can be used for classification or regression problems and can be used for any supervised learning algorithm.
- **Train Dataset**: Used to fit the machine learning model.
- **Test Dataset**: Used to evaluate the fit machine learning model.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)

# split again, and we should see the same split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
```

Example

```
>>> x
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12],
       [13, 14],
       [15, 16],
       [17, 18],
       [19, 20],
       [21, 22],
       [23, 24]])
>>> y
array([0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0])
```

```
>>> x_train, x_test, y_train, y_test = train_test_split(x, y)
>>> x_train
array([[15, 16],
       [21, 22],
       [11, 12],
       [17, 18],
       [13, 14],
       [ 9, 10],
       [ 1,  2],
       [ 3,  4],
       [19, 20]])
>>> x_test
array([[ 5,  6],
       [ 7,  8],
       [23, 24]])
>>> y_train
array([1, 1, 0, 1, 0, 1, 0, 1, 0])
>>> y_test
array([1, 0, 0])
```

You probably got different results from what you see here. This is because dataset splitting is random by default.

Split the data into Train and Test datasets

```
from sklearn.model_selection import train_test_split
y = continous_var_data['Customer Lifetime Value']
X = continous_var_data.drop(columns=['Customer Lifetime Value'])
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the shapes to confirm
print(f"Shape of X_train: {X_train.shape}")
print(f"Shape of X_test: {X_test.shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of y_test: {y_test.shape}")
```

```
Shape of X_train: (7307, 7)
Shape of X_test: (1827, 7)
Shape of y_train: (7307,)
Shape of y_test: (1827,)
```

Split the data into Train and Test datasets

Every time we change the random state, different observation gets selected into the training and testing.

```
continous_var_data=data.select_dtypes(include='number')

from sklearn.model_selection import train_test_split
y = continous_var_data['Customer Lifetime Value']
X = continous_var_data.drop(columns=['Customer Lifetime Value'])
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
print("shape of X_train ",X_train.shape)
print("shape of X_test ",X_test.shape)
print("Average of CLV in training data ",y_train.mean())
print("Average of CLV in testing data ",y_test.mean())
```

```
shape of X_train  (7307, 7)
shape of X_test  (1827, 7)
Average of CLV in training data  8022.789393969208
Average of CLV in testing data  7933.554568581828
```

If the averaging value of the dependent variable differs significantly between training and testing, the model does not have a fair opportunity of learning what it can from the data.

Fitting the simple regressionn model

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

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

```
reg.intercept_
```

```
193.6434424279978
```

```
reg.coef_
```

Y=a*x+b

Y=coef*x+intercept

Linear Regression

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(X_train, y_train)
acc_log_train = round(reg.score(X_train, y_train)*100,2)
acc_log_test = round(reg.score(X_test, y_test)*100,2)
print("Training Accuracy:%{}".format(acc_log_train))
print("Testing Accuracy:%{}".format(acc_log_test))
```

```
Training Accuracy:%16.2
Testing Accuracy:%15.3
```

It is not good

END