Add blockquote

Forecasting Sales and Visualizing Ad Campaign Impact on Social Media Platforms using Machine Learning

↳ 3 cells hidden

Importing Libraries

```
import numpy as np
import pandas as pd
```

Importing the required libraries

Loading Data

Loading the data using read_csv

```
from google.colab import files
uploaded = files.upload()
```

Choose Files ad_campai…analysis.csv

- **ad_campaign_analysis.csv**(text/csv) - 60522 bytes, last modified: 9/29/2024 - 100% done

```
Saving ad_campaign_analysis.csv to ad_campaign_analysis.csv
```

```
data_frame=pd.read_csv("ad_campaign_analysis.csv")
```

Printing the first 5 columns using data.head()

```
data_frame.head()
```

	ad_id	xyz_campaign_id	fb_campaign_id	age	gender	interest	Impressions	Clicks	Spent	Total_Conversion	Approved_Conversion
0	708746	916	103916	30-34	M	15	7350	1	1.43	2	1
1	708749	916	103917	30-34	M	16	17861	2	1.82	2	0
2	708771	916	103920	30-34	M	20	693	0	0.00	1	0

Next steps: Generate code with `data_frame` View recommended plots New interactive sheet

Checking and handling for null values

```
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ad_id                1143 non-null   int64
 1   xyz_campaign_id      1143 non-null   int64
 2   fb_campaign_id       1143 non-null   int64
 3   age                  1143 non-null   object
 4   gender               1143 non-null   object
```

```
 5   interest             1143 non-null   int64
 6   Impressions          1143 non-null   int64
 7   Clicks               1143 non-null   int64
 8   Spent                1143 non-null   float64
 9   Total_Conversion     1143 non-null   int64
 10  Approved_Conversion  1143 non-null   int64
dtypes: float64(1), int64(8), object(2)
memory usage: 98.4+ KB
```

Exploratory Data Analysis

```
data_frame.shape
```

```
(1143, 11)
```

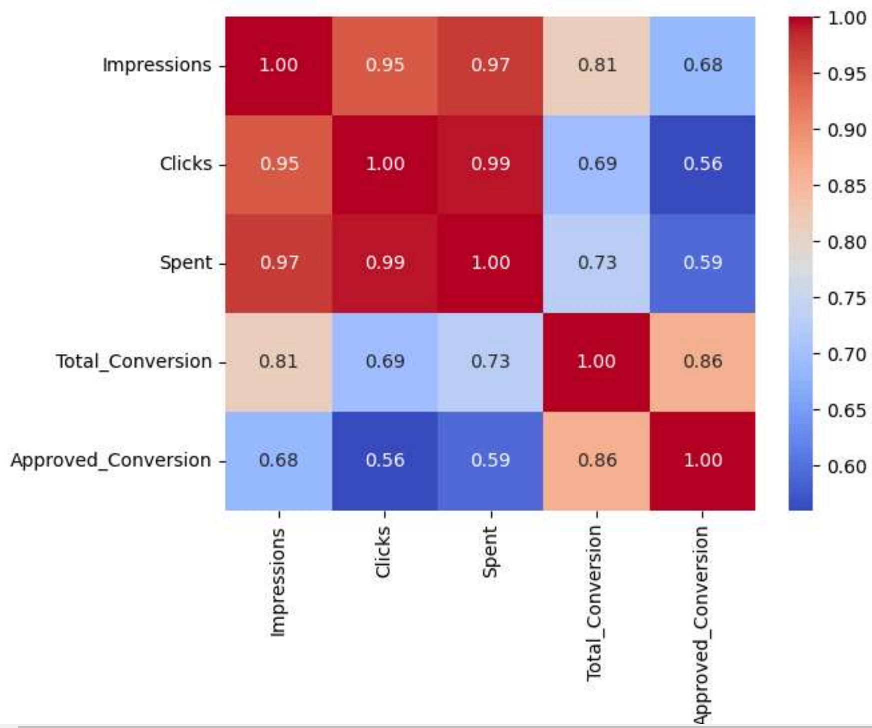
```
data_frame.describe()
```

	ad_id	xyz_campaign_id	fb_campaign_id	interest	Impressions	Clicks	Spent	Total_Conversion	Approved_Conve
count	1.143000e+03	1143.000000	1143.000000	1143.000000	1.143000e+03	1143.000000	1143.000000	1143.000000	1143.(
mean	9.872611e+05	1067.382327	133783.989501	32.766404	1.867321e+05	33.390201	51.360656	2.855643	0.9
std	1.939928e+05	121.629393	20500.308622	26.952131	3.127622e+05	56.892438	86.908418	4.483593	1.7
min	7.087460e+05	916.000000	103916.000000	2.000000	8.700000e+01	0.000000	0.000000	0.000000	0.(
25%	7.776325e+05	936.000000	115716.000000	16.000000	6.503500e+03	1.000000	1.480000	1.000000	0.(
50%	1.121185e+06	1178.000000	144549.000000	25.000000	5.150900e+04	8.000000	12.370000	1.000000	1.(
75%	1.121804e+06	1178.000000	144657.500000	31.000000	2.217690e+05	37.500000	60.025000	3.000000	1.(
max	1.314415e+06	1178.000000	179982.000000	114.000000	3.052003e+06	421.000000	639.949998	60.000000	21.(

```
# Importing the Libararies for visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

Generating Correlation Matrix

```
corr=sns.heatmap(data_frame[["Impressions","Clicks","Spent","Total_Conversion","Approved_Conversion"]].corr(),annot=True ,fmt=".2f", cmap="c
```



It can be understood that impressions and total conversion are correlated with the approved conversion than with the clicks and spent.

Ad-Campaigns

```
data_frame["xyz_campaign_id"].unique()
```

```
array([ 916,  936, 1178])
```

Three different types of ad campaigns can be seen here for xyz company. Replacing these ids with CAMPAIGN_A, CAMPAIGN_B and CAMPAIGN_C

```
data_frame["xyz_campaign_id"].replace({916:"CAMPAIGN_A",936:"CAMPAIGN_B",1178:"CAMPAIGN_C"}, inplace=True)
```

```
<ipython-input-12-116cbd7a5af0>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignm
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting valu

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].me

  data_frame["xyz_campaign_id"].replace({916:"CAMPAIGN_A",936:"CAMPAIGN_B",1178:"CAMPAIGN_C"}, inplace=True)
```

```
data_frame.head()
```

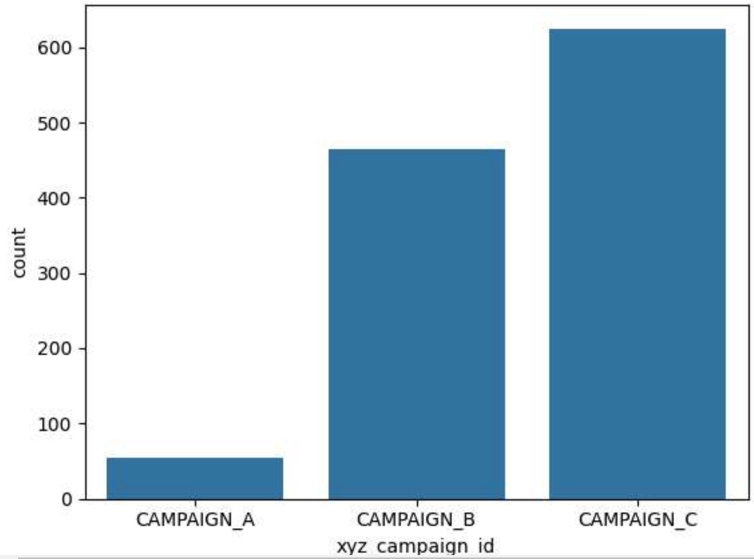
	ad_id	xyz_campaign_id	fb_campaign_id	age	gender	interest	Impressions	Clicks	Spent	Total_Conversion	Approved_Conversion
0	708746	CAMPAIGN_A	103916	30-34	M	15	7350	1	1.43	2	1
1	708749	CAMPAIGN_A	103917	30-34	M	16	17861	2	1.82	2	0
2	708771	CAMPAIGN_A	103920	30-34	M	20	693	0	0.00	1	0

Next steps: Generate code with data_frame View recommended plots New interactive sheet

The campaign ids are changed.

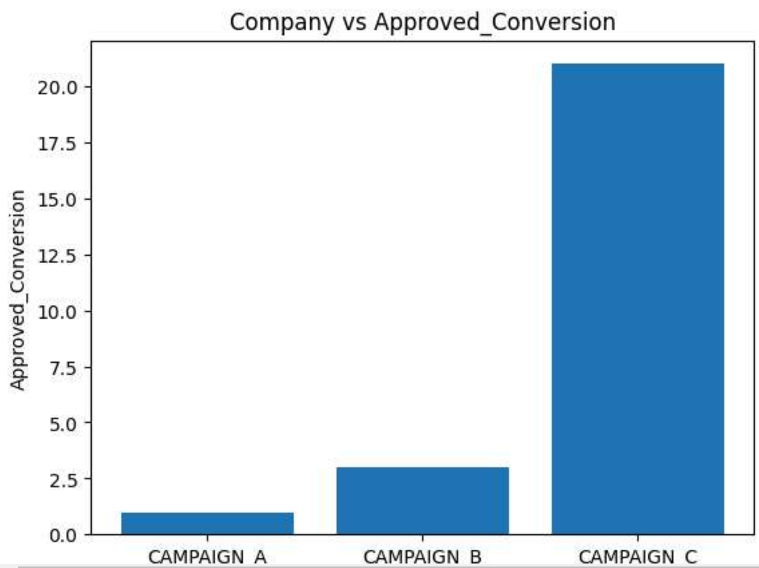
Performing Data Vizualization

```
# visualizing count plot on single categorical variable
sns.countplot(x ='xyz_campaign_id', data = data_frame)
# Generating the plot
plt.show()
```



CAMPAIGN_C has more ads compared to CAMPAIGN_A and CAMPAIGN_B.

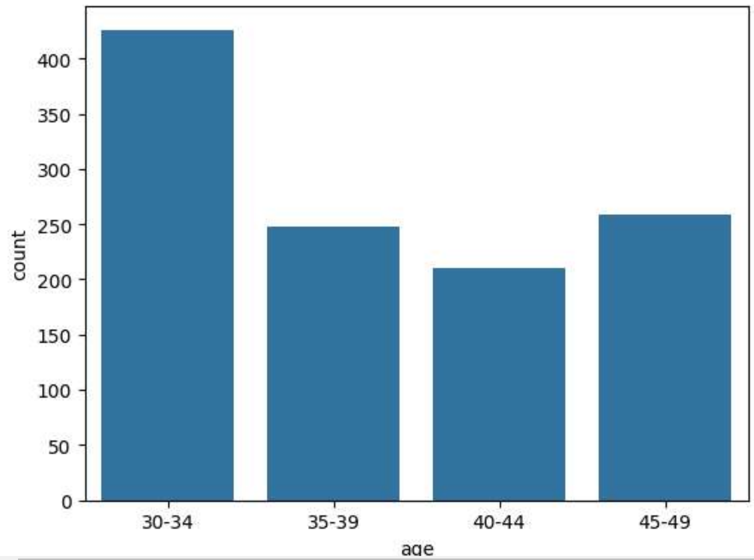
```
#Company vs Approved_Conversion
# Generating bar plot
plt.bar(data_frame["xyz_campaign_id"], data_frame["Approved_Conversion"])
plt.ylabel("Approved_Conversion")
plt.title("Company vs Approved_Conversion")
plt.show()
```



CAMPAIGN_C has better approved conversion count than CAMPAIGN_A and CAMPAIGN_B concluding that most people bought products from CAMPAIGN_C.

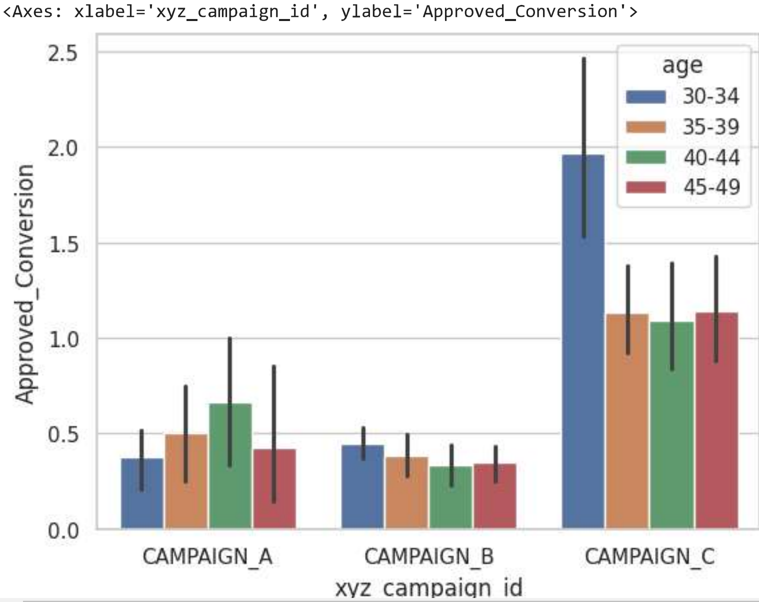
Age

```
# count plot on single categorical variable
sns.countplot(x ='age', data = data_frame)
# Generating the plot
plt.show()
```




```
import seaborn as sns
sns.set(style="whitegrid")
tips = sns.load_dataset("tips")
sns.barplot(x=data_frame["xyz_campaign_id"], y=data_frame["Approved_Conversion"], hue=data_frame["age"], data=tips)
```

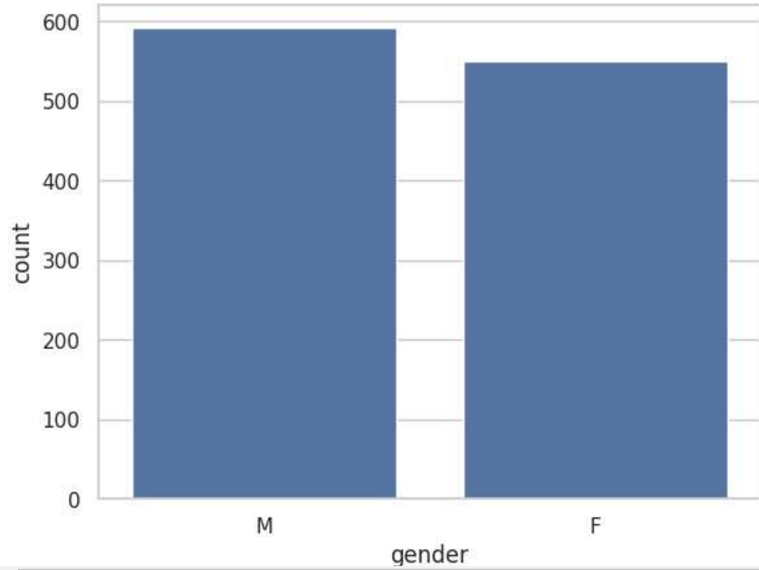
<Axes: xlabel='xyz_campaign_id', ylabel='Approved_Conversion'>



30-34 age group showed more interest in CAMPAIGN_B and CAMPAIGN_C while 40-44 age group showed more interest for CAMPAIGN_A.

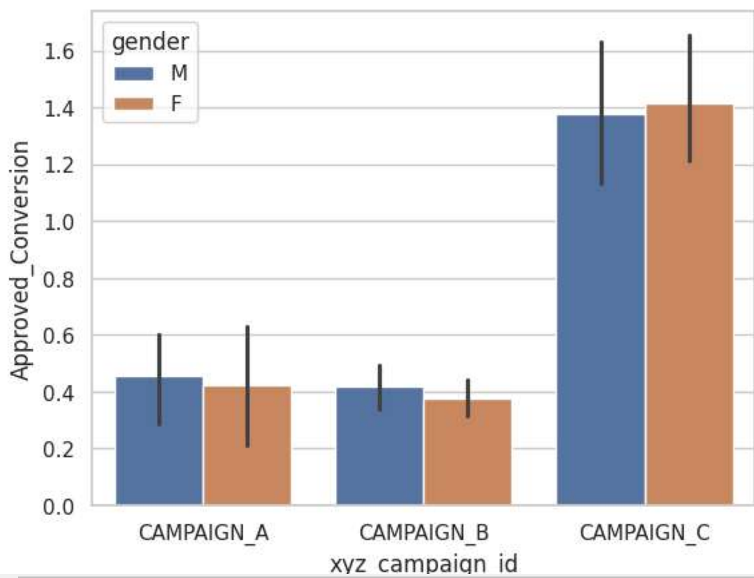
Gender

```
# count plot on single categorical variable
sns.countplot(x ='gender', data = data_frame)
# Generating the plot
plt.show()
```




```
import seaborn as sns
sns.set(style="whitegrid")
tips = sns.load_dataset("tips")
sns.barplot(x=data_frame["xyz_campaign_id"], y=data_frame["Approved_Conversion"], hue=data_frame["gender"], data=tips)
```

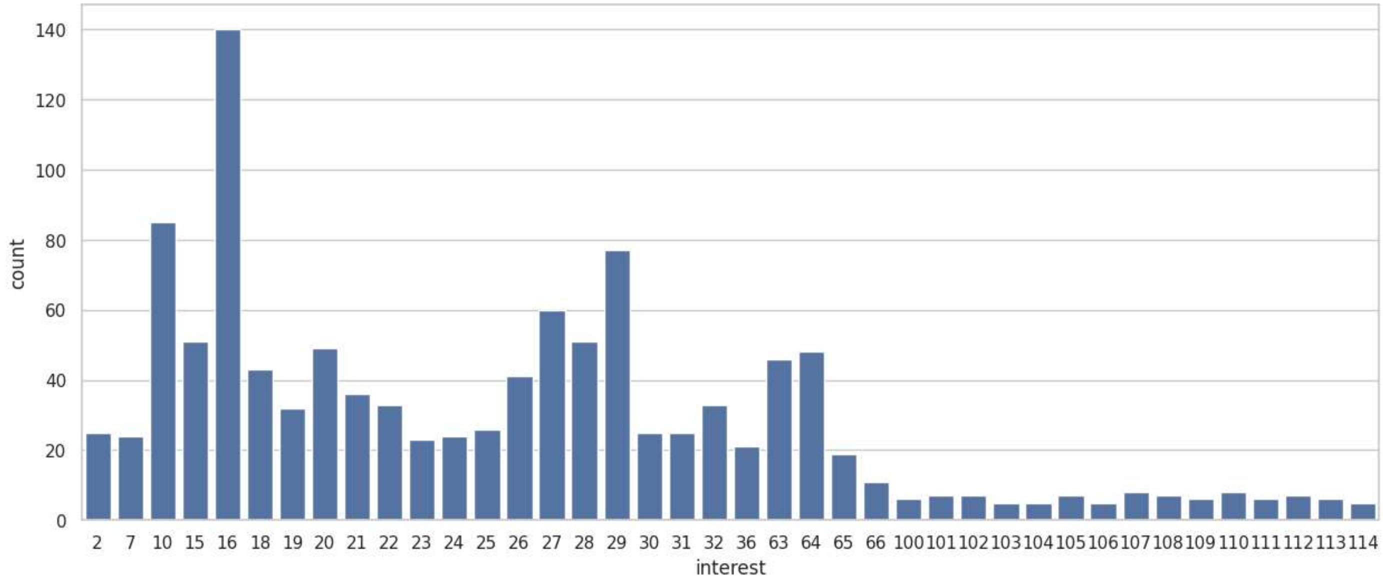
```
<Axes: xlabel='xyz_campaign_id', ylabel='Approved_Conversion'>
```



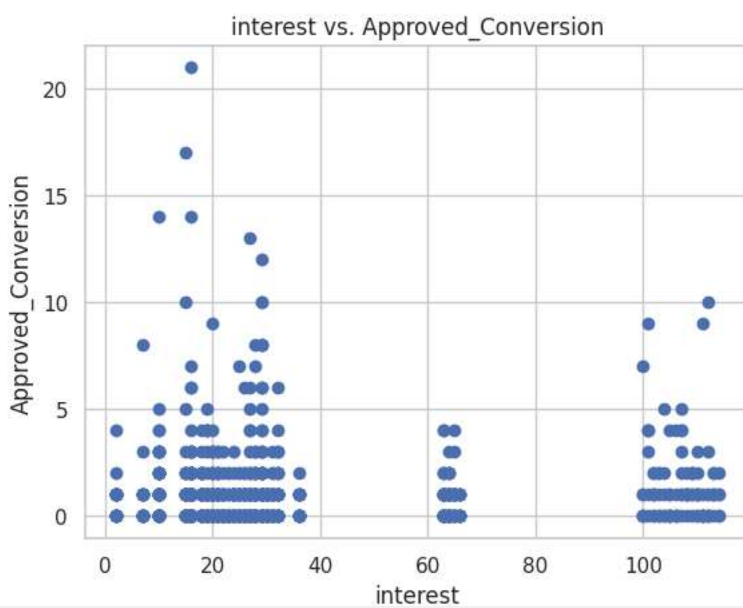
Almost both of the genders showed same interests in all the three campaigns.

Interest

```
# count plot on single categorical variable
fig_dims = (15,6)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(x ='interest', data = data_frame)
# Generating the plot
plt.show()
```

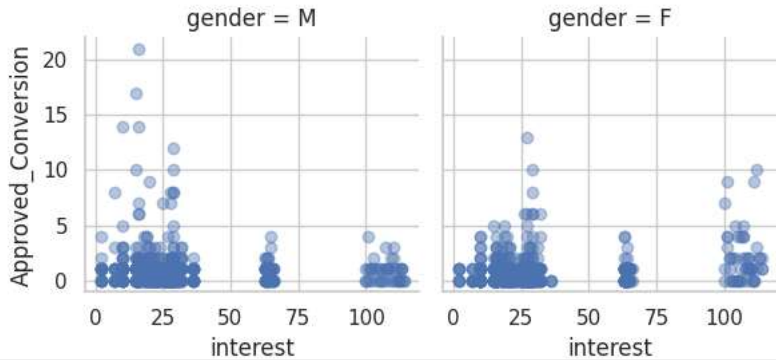



```
plt.scatter(data_frame["interest"], data_frame["Approved_Conversion"])
plt.title("interest vs. Approved_Conversion")
plt.xlabel("interest")
plt.ylabel("Approved_Conversion")
plt.show()
```

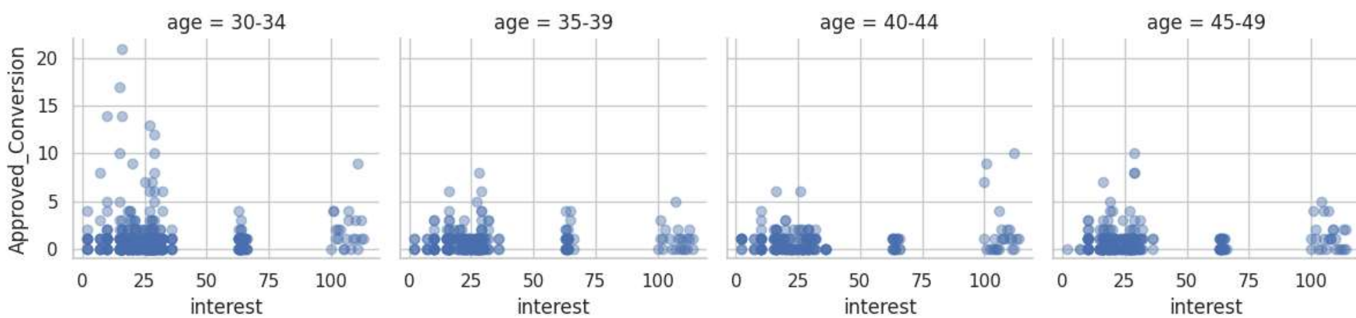


It's interesting to see that even while there were less people showing interest after 100, there was a spike in the number of people who made a purchase. The output of the distribution is as anticipated.

```
p = sns.FacetGrid(data_frame, col="gender")
p.map(plt.scatter, "interest", "Approved_Conversion", alpha=.4)
p.add_legend();
```

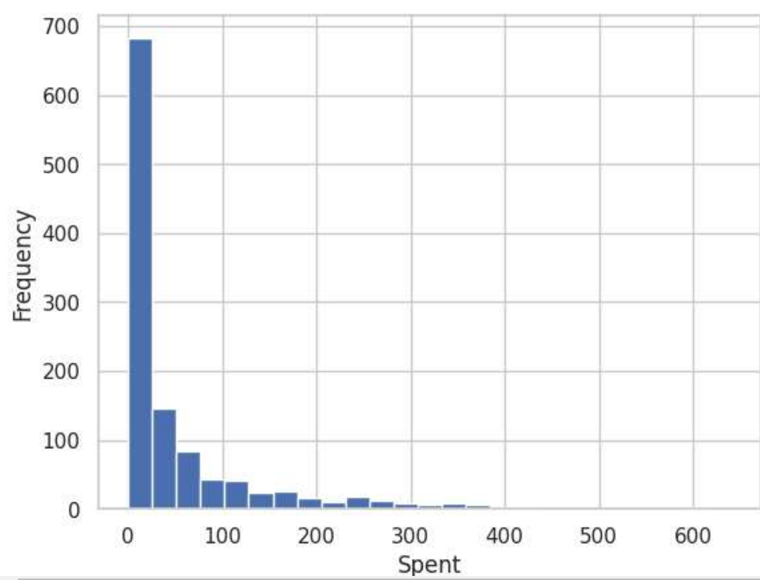



```
p = sns.FacetGrid(data_frame, col="age")
p.map(plt.scatter, "interest", "Approved_Conversion", alpha=.4)
p.add_legend();
```

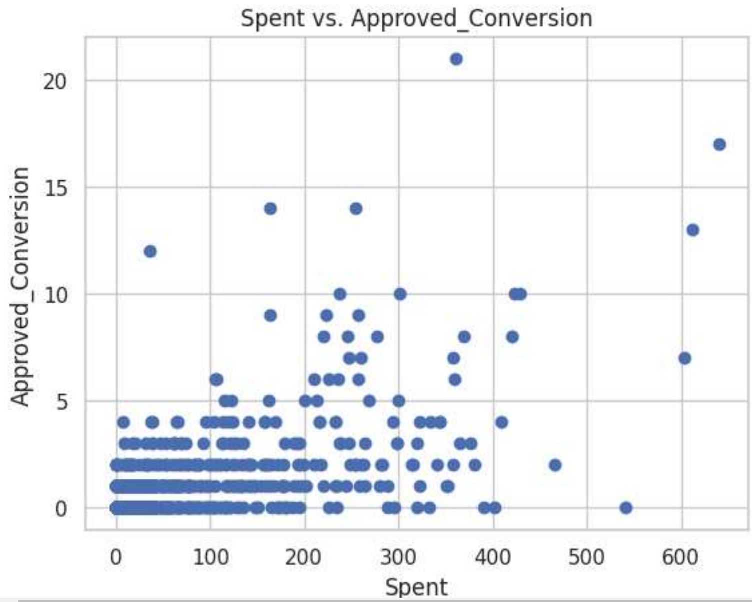


Spent

```
plt.hist(data_frame['Spent'], bins = 25)
plt.xlabel("Spent")
plt.ylabel("Frequency")
plt.show()
```

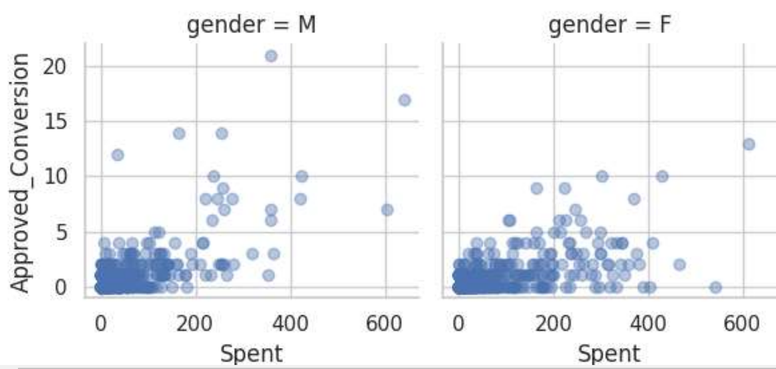



```
# spent vs approved conversion
plt.scatter(data_frame["Spent"], data_frame["Approved_Conversion"])
plt.title("Spent vs. Approved_Conversion")
plt.xlabel("Spent")
plt.ylabel("Approved_Conversion")
plt.show()
```



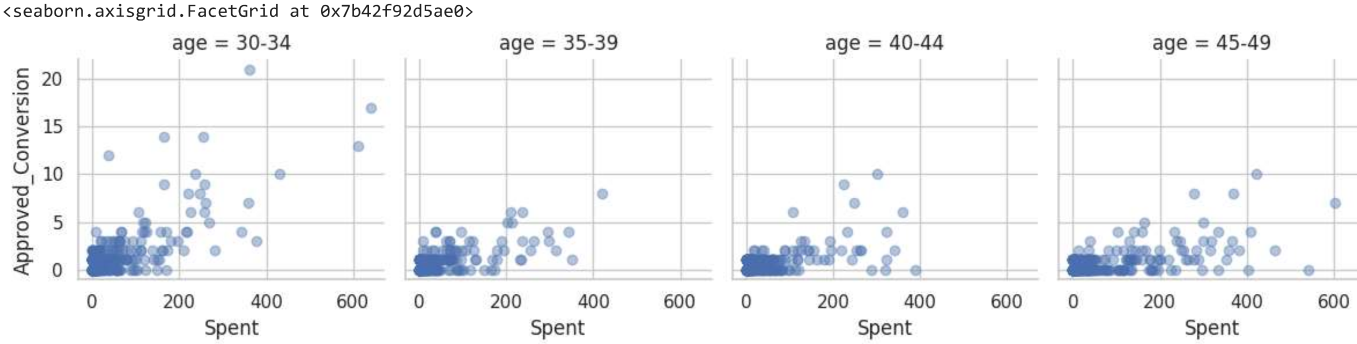
No of products bought is directly proportional to the spents

```
p = sns.FacetGrid(data_frame, col="gender")
p.map(plt.scatter, "Spent", "Approved_Conversion", alpha=.4)
p.add_legend();
```



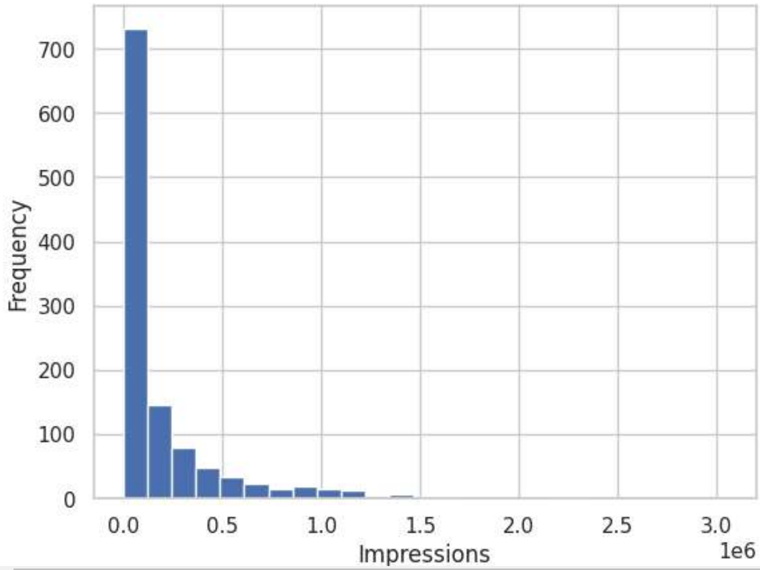

```
# Creating a FacetGrid with the column variable as age
p = sns.FacetGrid(data_frame, col="age")

# mapping Scatter plot for Spent vs Approved Conversion for each age group
p.map(plt.scatter, "Spent", "Approved_Conversion", alpha=.4)

# Giving legend for plot
p.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x7b42f92d5ae0>
```

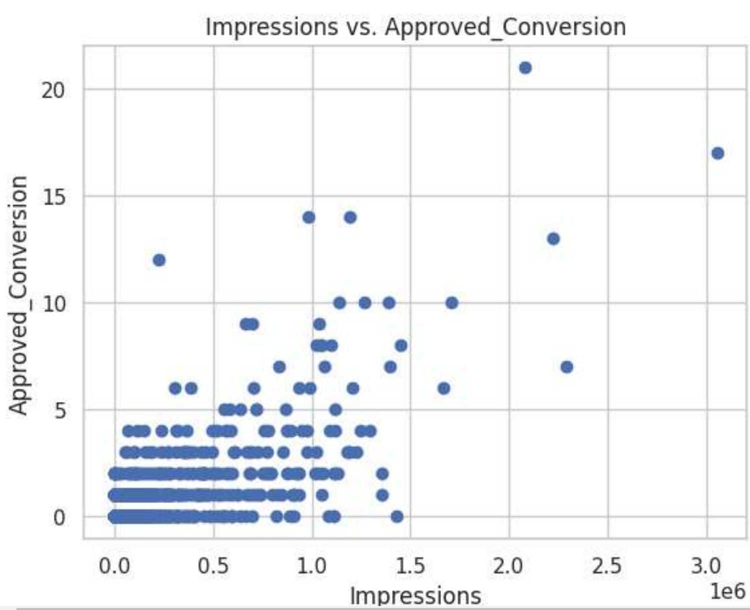


Impressions

```
plt.hist(data_frame['Impressions'], bins = 25)
plt.xlabel("Impressions")
plt.ylabel("Frequency")
plt.show()
```




```
plt.scatter(data_frame["Impressions"], data_frame["Approved_Conversion"])
plt.title("Impressions vs. Approved_Conversion")
plt.xlabel("Impressions")
plt.ylabel("Approved_Conversion")
plt.show()
```



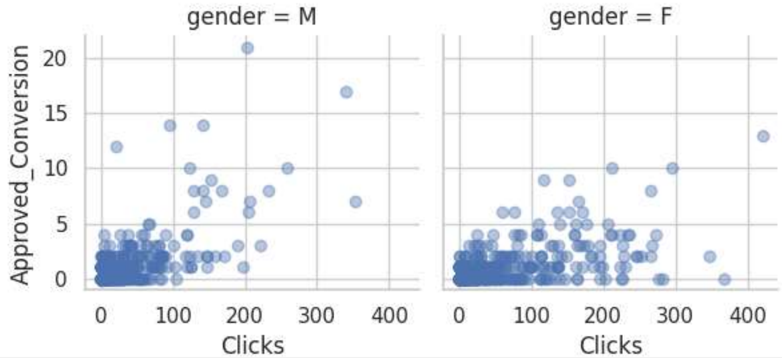
It can be observed that after some point in impressions the approved conversions is increased.

Checking who bought the product actually

Clicking on the ad ?

Checking who actually bought the product after clicking on the ad.

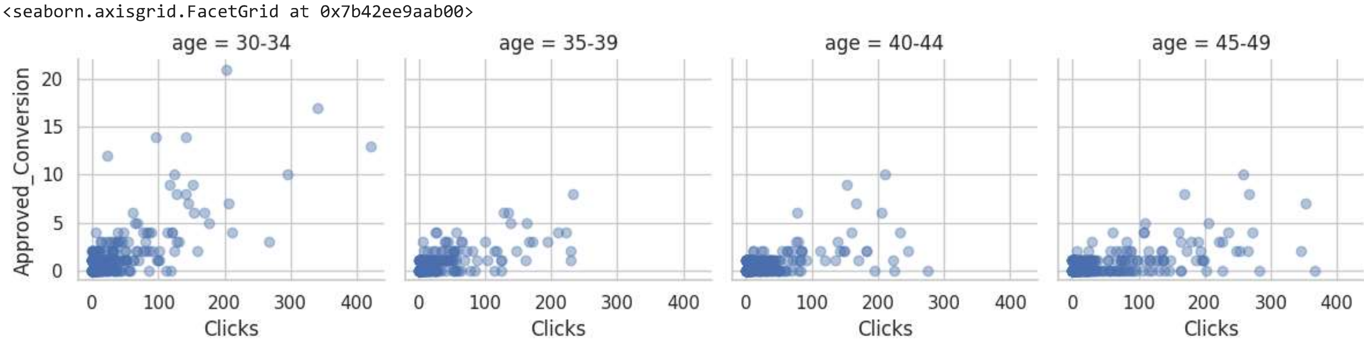
```
p = sns.FacetGrid(data_frame, col="gender")
p.map(plt.scatter, "Clicks", "Approved_Conversion", alpha=.4)
p.add_legend();
```



Men clicked on the ad more than women but women ended up buying the product more than men.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Creating a FacetGrid with the column variable as age
p = sns.FacetGrid(data_frame, col="age")

# Mapping scatter plot of Clicks vs Approved_Conversion for each age group
p.map(plt.scatter, "Clicks", "Approved_Conversion", alpha=.4)

# Giving legend for plot
p.add_legend()
```

<seaborn.axisgrid.FacetGrid at 0x7b42ee9aab00>

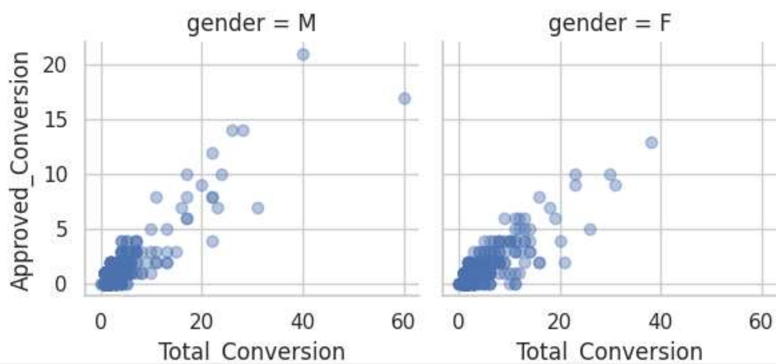


30-34 age group people tend to buy the product after clicking the ad.

Enquiring about the product?

Checking who actually bought the product after enquiring about the ad.

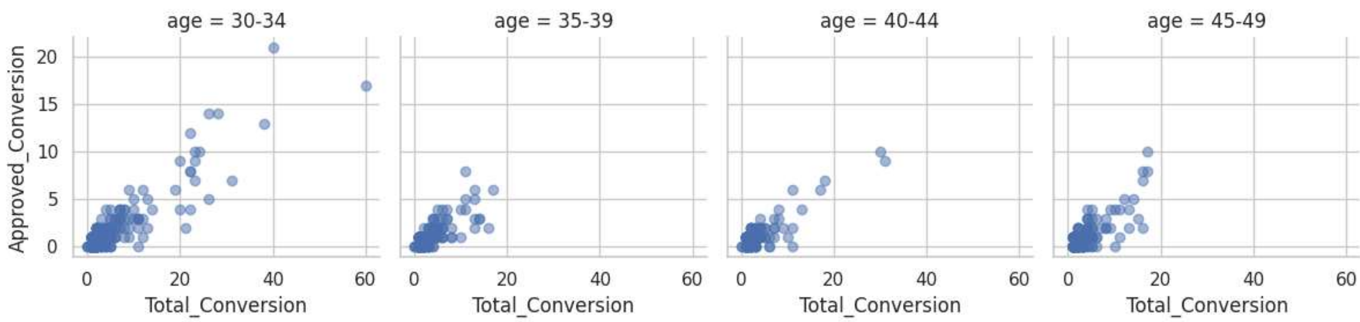
```
p = sns.FacetGrid(data_frame, col="gender")
p.map(plt.scatter, "Total_Conversion", "Approved_Conversion", alpha=.4)
p.add_legend();
```



Men enquired about the ad more than women but women ended up buying the product more than men.

```
p = sns.FacetGrid(data_frame, col="age")
p.map(plt.scatter, "Total_Conversion", "Approved_Conversion",alpha=.5)
p.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x7b42ef043df0>
```



30-34 age group people tend to buy the product after enquiring about the ad.

Zoom-in the campaign which has the most approved conversion (Campaign_C)

```
c_a=[]
c_b=[]
c_c=[]
for i,j,k in zip(data_frame.xyz_campaign_id, data_frame.fb_campaign_id, data_frame.Approved_Conversion):
    if i=="CAMPAIGN_C":
        c_a.append(i),c_b.append(j),c_c.append(k)
```

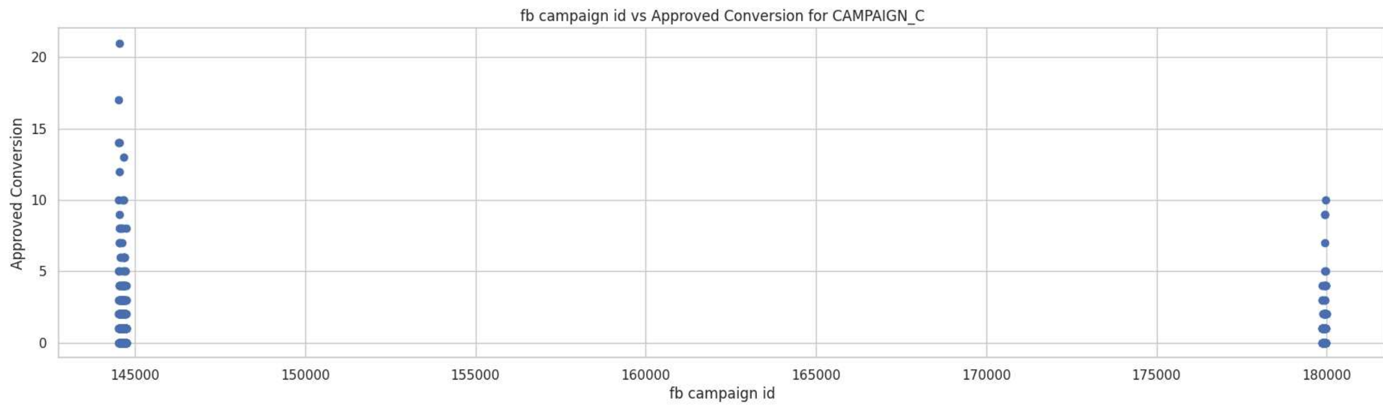
```
z={'campaign_name':c_a, 'fb_campaign_id':c_b, 'Approved_Conversion':c_c}
c_campaign=pd.DataFrame(z)
c_campaign.head()
```

	campaign_name	fb_campaign_id	Approved_Conversion
0	CAMPAIGN_C	144531	14
1	CAMPAIGN_C	144531	5
2	CAMPAIGN_C	144531	1
3	CAMPAIGN_C	144531	2
4	CAMPAIGN_C	144531	2

Next steps: Generate code with c_campaign | View recommended plots | New interactive sheet

Distribution of the fb campaign id along with the Approved Conversion for CAMPAIGN_C

```
plt.figure(figsize=(20,5))
plt.scatter(c_campaign["fb_campaign_id"], c_campaign["Approved_Conversion"])
plt.title("fb campaign id vs Approved Conversion for CAMPAIGN_C")
plt.xlabel("fb campaign id")
plt.ylabel("Approved Conversion")
plt.show()
```



The fb campaign ids are approximately 145000 which has more approved conversion and for campaign c it was around 180000.

Conclusion

Correlations analysis: Impressions and total conversion are correlated with the approved conversion than with the clicks and spent.

Campaign_C: CAMPAIGN_C has more ads. CAMPAIGN_C has better approved conversion count than CAMPAIGN_A and CAMPAIGN_B concluding that most people bought products from CAMPAIGN_C.

age_group: 30-34 age group showed more interest in CAMPAIGN_B and CAMPAIGN_C while 40-44 age group showed more interest for CAMPAIGN_A.

gender: Almost both of the genders showed same interests in all the three campaigns.

interest: It's interesting to see that even while there were less people showing interest after 100, there was a spike in the number of people who made a purchase. The output of the distribution is as anticipated.

money spent: No of products bought is directly proportional to the spent.

Product bought after clicking the ad: Men clicked on the ad more than women but women ended up buying the product more than men. 30-34 age group people tend to buy the product after clicking the ad.

Product bought after enquiring the ad: Men enquired about the ad more than women but women ended up buying the product more than men. 30-34 age group people tend to buy the product after enquiring about the ad.

Conclusion:

The fb campaign ids are approximately 145000 which has more approved conversion and for campaign c it was around 180000.

Modelling to train the data

Actual ids are assigned to the xyz_campaign_id for the modelling purpose

```
data_frame["xyz_campaign_id"].replace({"CAMPAIGN_A":916 ,"CAMPAIGN_B":936 ,"CAMPAIGN_C":1178}, inplace=True)
```

```
<ipython-input-37-ae595f44745a>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future versic
  data_frame["xyz_campaign_id"].replace({"CAMPAIGN_A":916 ,"CAMPAIGN_B":936 ,"CAMPAIGN_C":1178}, inplace=True)
```

Encoding of the 'gender' and 'age' (labels) for modelling

```
#encoding the genders
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
encoder.fit(data_frame["gender"])
data_frame["gender"]=encoder.transform(data_frame["gender"])
print(data_frame["gender"])
```

```
0       1
1       1
2       1
3       1
4       1
       ..
1138    0
1139    0
1140    0
1141    0
1142    0
Name: gender, Length: 1143, dtype: int64
```

```
#encoding the age
encoder.fit(data_frame["age"])
data_frame["age"]=encoder.transform(data_frame["age"])
print(data_frame["age"])
```

```
0       0
1       0
2       0
3       0
4       0
       ..
1138    3
1139    3
1140    3
1141    3
1142    3
Name: age, Length: 1143, dtype: int64
```

```
data_frame.head()
```

	ad_id	xyz_campaign_id	fb_campaign_id	age	gender	interest	Impressions	Clicks	Spent	Total_Conversion	Approved_Conversion
0	708746	916	103916	0	1	15	7350	1	1.43	2	1
1	708749	916	103917	0	1	16	17861	2	1.82	2	0
2	708771	916	103920	0	1	20	693	0	0.00	1	0
3	708815	916	103928	0	1	28	4259	1	1.25	1	0
4	708818	916	103928	0	1	28	4133	1	1.29	1	1

Next steps: Generate code with data_frame | View recommended plots | New interactive sheet

Deleting "Approved_Conversion" and "Total_Conversion" from dataset

```
x=np.array(data_frame.drop(labels=["Approved_Conversion","Total_Conversion"], axis=1))
y=np.array(data_frame["Total_Conversion"])
```

```
x
```

```
array([[7.08746000e+05, 9.16000000e+02, 1.03916000e+05, ...,
         7.35000000e+03, 1.00000000e+00, 1.42999995e+00],
       [7.08749000e+05, 9.16000000e+02, 1.03917000e+05, ...,
         1.78610000e+04, 2.00000000e+00, 1.82000002e+00],
       [7.08771000e+05, 9.16000000e+02, 1.03920000e+05, ...,
         6.93000000e+02, 0.00000000e+00, 0.00000000e+00],
       ...,
       [1.31441200e+06, 1.17800000e+03, 1.79979000e+05, ...,
         1.51531000e+05, 2.80000000e+01, 4.02899995e+01],
       [1.31441400e+06, 1.17800000e+03, 1.79981000e+05, ...,
         7.90253000e+05, 1.35000000e+02, 1.98710001e+02],
       [1.31441500e+06, 1.17800000e+03, 1.79982000e+05, ...,
         5.13161000e+05, 1.14000000e+02, 1.65609999e+02]])
```

```
y
```

```
array([2, 2, 1, ..., 2, 8, 5])
```

```
y=y.reshape(len(y),1)
y
```

```
array([[2],
       [2],
       [1],
       ...,
       [2],
       [8],
       [5]])
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
x_sc= StandardScaler()
x = x_sc.fit_transform(x)
```

splitting Data into testset and trainset

```
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)
```

Random Forest Classifier to predict Total_Conversion

```
from sklearn.ensemble import RandomForestClassifier
random_fr = RandomForestClassifier(n_estimators = 10, random_state = 0)
random_fr.fit(x_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1473: DataConversionWarning: A column-vector y was passed when a 1d array was ex
  return fit_method(estimator, *args, **kwargs)
          RandomForestClassifier
RandomForestClassifier(n_estimators=10, random_state=0)
```

Predicting Total Conversion in test_set

```
# Make predictions on the test set
y_pred = random_fr.predict(x_test)
```

```
y_pred
```

```
array([ 1,  4,  1,  2,  1,  4,  1,  1,  1,  1,  1, 13,  2,  1,  2,  1,  1,
        2,  1,  2,  4,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  2,  1,  1,  1,  1,  2,  3,  1,  1,  1,  1,  3,  0,
        1,  1,  2,  1,  1,  2,  5,  2,  1,  1,  2,  1,  1,  3,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  3,  1,
        1,  1,  1, 24,  1,  1,  2,  1,  4,  1,  1,  4,  1,  1,  1,  3,  2,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  2,  1,  1,  1,  1,  2,
        1,  3,  1,  1,  1, 17,  4,  3,  1,  3,  1,  1,  1,  2,  1,  1,  1,
        1,  1,  1,  6,  1,  1,  1,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1, 22,  1,  3,  1,  2,  1,  4,  1,  1,  1, 16,  2,  1,  1,  1,
        3, 22,  5,  3,  1,  2,  1,  1,  1, 11,  1,  1,  1, 11,  1,  1,  2,
        1,  1,  1,  1,  1,  1,  1,  1,  6,  3,  3,  1,  2,  1,  1,  1,  1,
        7,  2,  1,  7,  8,  1,  1,  1,  1,  2,  1,  2,  1,  1,  1,  1,  5,
        1,  1,  3,  1,  1,  1,  1,  2,  1,  6,  1,  3,  1,  1,  1,  5,  3,
        1,  1, 31,  1,  5,  1,  1,  3,  2,  3,  2,  1,  1,  2,  1,  1,  1,
        1,  1,  1,  2,  1, 16, 11,  3,  1,  1, 24,  1,  2,  1,  2,  1,  1,
        1,  3,  1,  1,  1,  1,  1,  1,  1,  4,  1,  1,  9,  1,  6,  1,  3,
        4,  6,  1,  1,  1,  1,  1,  1,  1,  2,  4,  4,  1,  2,  1,  1,  1,
        5,  1,  2,  1,  1,  4,  1,  1,  1,  2,  1,  6,  1,  7,  1,  1,  1,
        2,  4,  2,  3,  1,  1,  2,  1,  2,  1,  9,  4,  1,  1,  1,  1,  1,
        1,  1,  3])
```

Evaluation

```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error,accuracy_score
mean_abs_error=mean_absolute_error(y_test, y_pred)
mean_sqr_error=mean_squared_error(y_test, y_pred)
r_mean_sqr_error=np.sqrt(mean_sqr_error)
r2_s=r2_score(y_test, y_pred)
a_s=accuracy_score(y_test, y_pred)
```

```
mean_abs_error
```

```
1.346938775510204
```

The mean absolute error achieved is 1.346.

```
#R-squred value
r2_s
```

```
0.5382662207706701
```

53.8% of the data will be fitting into the classifier model as the r2 score is achieved as 0.538.

```
a_s
```

```
0.5685131195335277
```

The accuracy is achieved as 0.568 which means the model is less accurate for the dataset.

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
import pandas as pd
```