```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings("ignore")
```

```
sales = pd.read_csv("KAG_conversion_data.csv")
sales.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

```
sales.describe()
```

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

```
sales.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
```

```
sales.shape
```

```
(1143, 11)
```

1. Check missing values

```
sales.isna().sum()
```

```
ad_id                0
xyz_campaign_id      0
fb_campaign_id       0
age                  0
gender               0
interest             0
Impressions          0
```

```
Clicks                 0
Spent                  0
Total_Conversion       0
Approved_Conversion    0
dtype: int64
```

There is no missing values in this dataset

2. Data Understanding

1. categorical features, and their unique values
2. check outliers, and remove outliers
3. find the features correlation matrix

Categorical Featuring:

```
# number of Campaigns
print(f"{len(sales['xyz_campaign_id'].unique())} unique values: {sales['xyz_campaign_id'].unique()}")
print(f"{len(sales['fb_campaign_id'].unique())} unique values")
```

```
3 unique values: [ 916  936 1178]
691 unique values
```

There are only 3 campaigns from xyz company, and we should change xyz_coampaign_id to categorical features later.

```
# other categorical features
categorical_features = sales.select_dtypes(include="object")
for col in categorical_features:
    print(f"Column Name: {col} unique values: {sales[col].unique()}")
    print(sales[col].value_counts())
```

```
Column Name: age unique values: ['30-34' '35-39' '40-44' '45-49']
30-34    426
45-49    259
35-39    248
40-44    210
Name: age, dtype: int64
Column Name: gender unique values: ['M' 'F']
M    592
F    551
Name: gender, dtype: int64
```

```
# check number of unique values for all columns
for col_name in sales.columns:
    print(f"{col_name} has {len(sales[col_name].unique())} values")
```

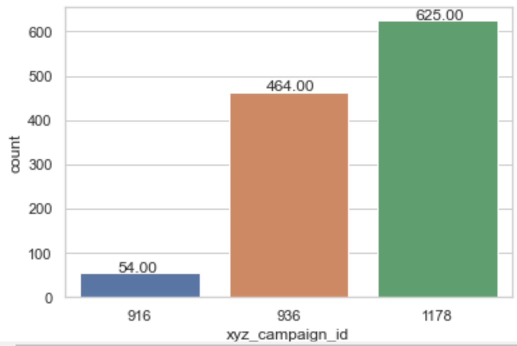
```
ad_id has 1143 values
xyz_campaign_id has 3 values
fb_campaign_id has 691 values
age has 4 values
gender has 2 values
interest has 40 values
Impressions has 1130 values
Clicks has 183 values
Spent has 869 values
Total_Conversion has 32 values
Approved_Conversion has 16 values
```

```
# Categorical features are "ad_id", "xyz_campaign_id", "fb_campaign_id", "age", "gender", "interest"
categories = ["ad_id", "xyz_campaign_id", "fb_campaign_id", "age", "gender", "interest"]
sales[categories] = sales[categories].astype('category')
sales.info()
```

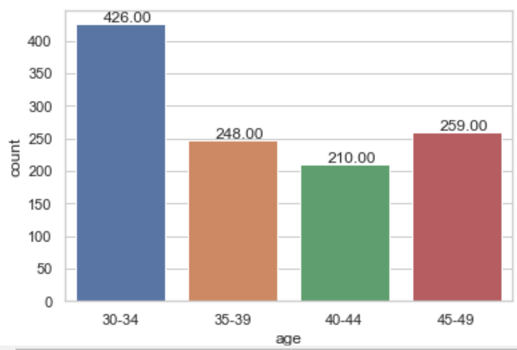
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ad_id                1143 non-null   category
 1   xyz_campaign_id      1143 non-null   category
 2   fb_campaign_id       1143 non-null   category
 3   age                  1143 non-null   category
 4   gender               1143 non-null   category
 5   interest             1143 non-null   category
 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: category(6), float64(1), int64(4)
memory usage: 130.0 KB
```

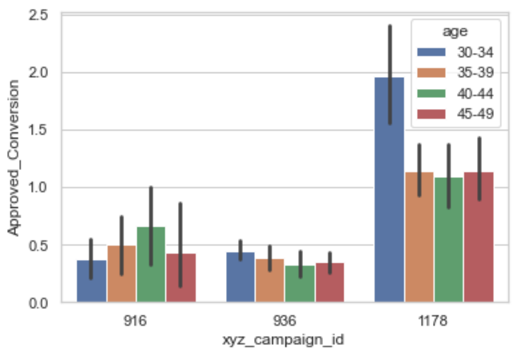
```
sns.set(style="whitegrid")
ax = sns.countplot(sales["xyz_campaign_id"])
for p in ax.patches:
    ax.annotate('{:.2f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+3))
```



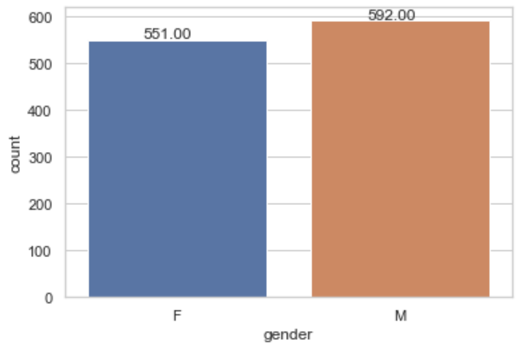

```
ax = sns.countplot(sales["age"])
for p in ax.patches:
    ax.annotate('{:.2f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+3))
```



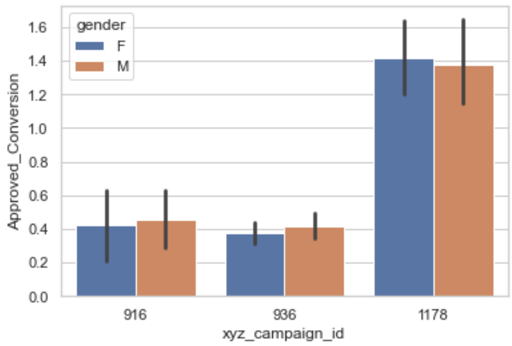

```
ax = sns.barplot(x = sales["xyz_campaign_id"], y = sales['Approved_Conversion'], hue = sales['age'])
```



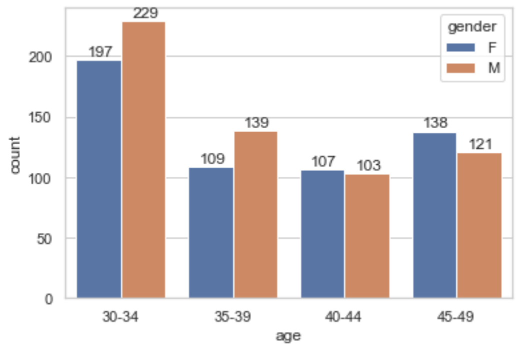

```
ax = sns.countplot(sales["gender"])
for p in ax.patches:
    ax.annotate('{:.2f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+3))
```




```
ax = sns.barplot(x = sales["xyz_campaign_id"], y = sales['Approved_Conversion'], hue = sales['gender'])
```

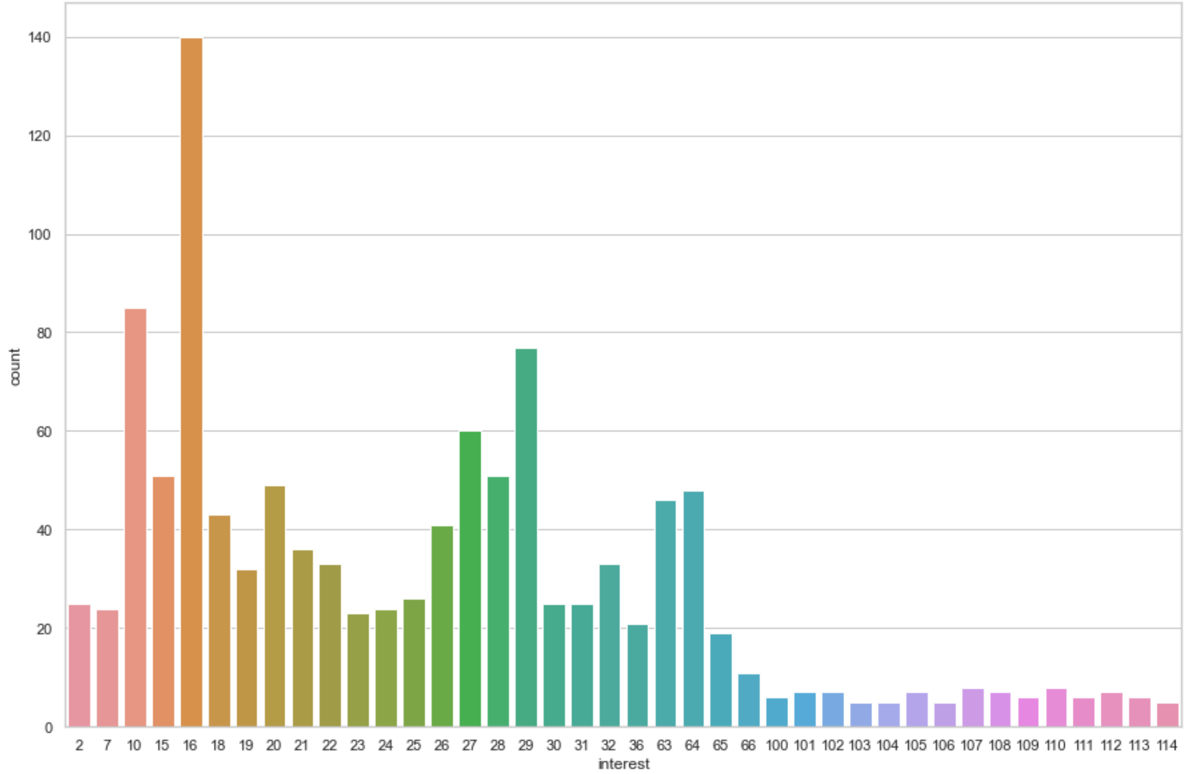



```
ax = sns.countplot(sales["age"],hue = sales['gender'] )
for p in ax.patches:
    ax.annotate(p.get_height(), (p.get_x()+0.1, p.get_height()+3))
```




```
plt.figure(figsize = (15, 10))
sns.countplot(sales["interest"])
```

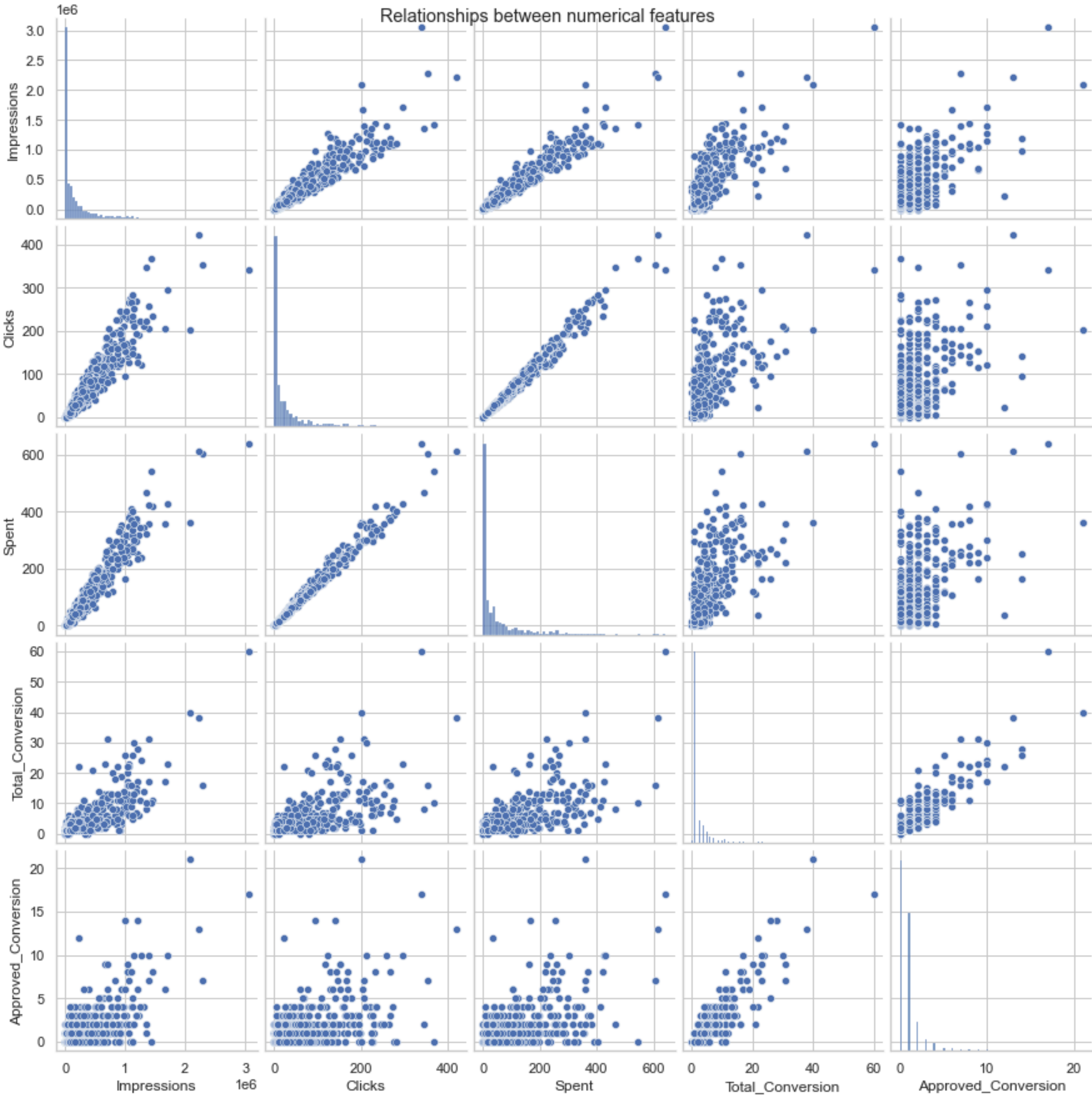
<AxesSubplot:xlabel='interest', ylabel='count'>



Numerical features relationships

```
numerical_features = sales.iloc[:,3:].select_dtypes(include=["float64","int64" ])
sns.pairplot(numerical_features) #since first 3 are
plt.suptitle("Relationships between numerical features")
```

```
Text(0.5, 0.98, 'Relationships between numerical features')
```



Check outliers

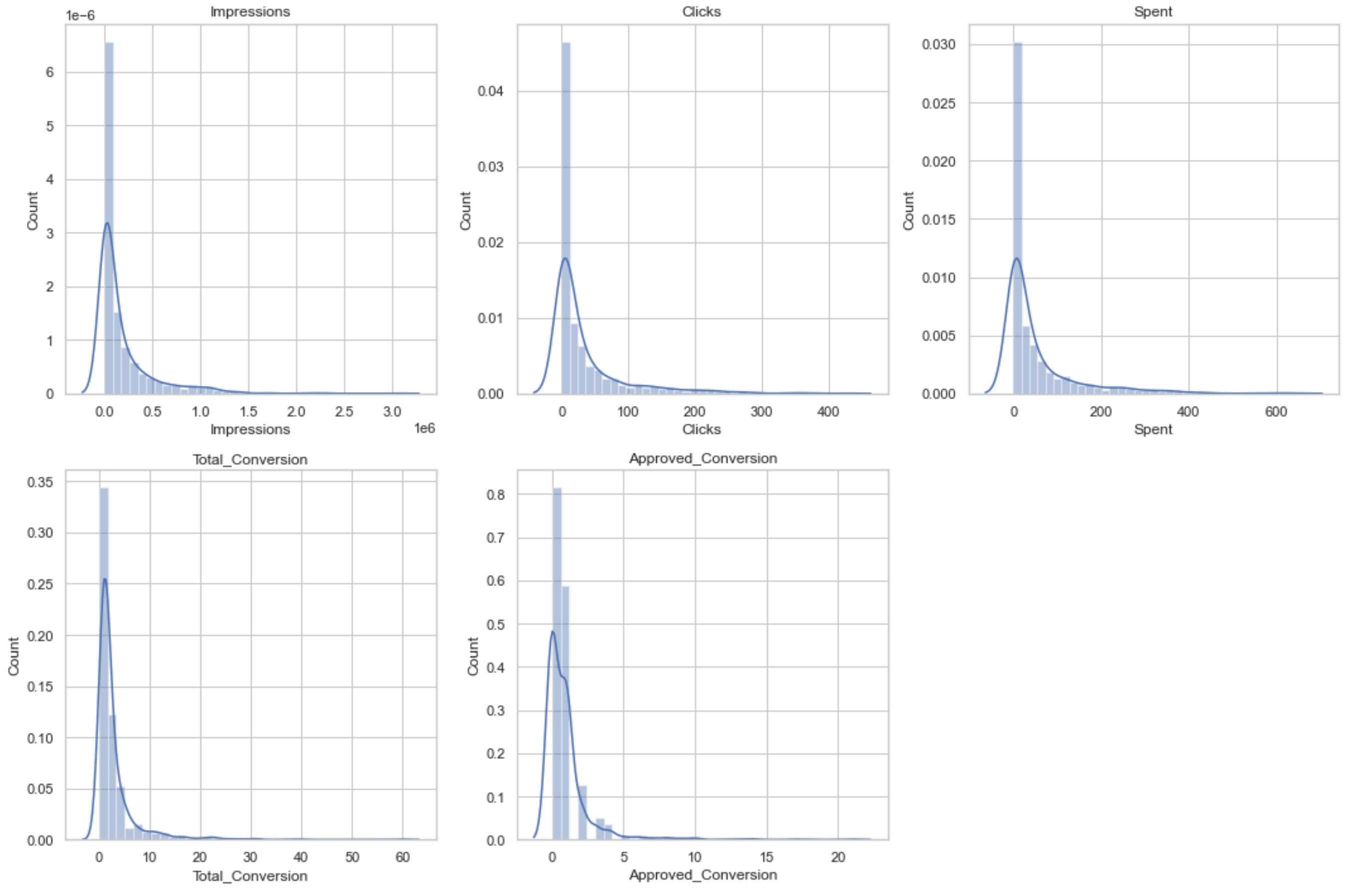
1. since first 3 columns are id, we start checking outliers from column 3
2. check zscore: >3 or <-3 is considered to be an outlier

```
# First check skewness of each column,
sales.iloc[:,3:].skew()
```

```
Impressions            3.010185
Clicks                 2.712187
Spent                  2.708867
Total_Conversion       5.095919
Approved_Conversion    4.837539
dtype: float64
```

when result is >0, towards the right hand side of distribution, and we will plot out the distribution

```
numerical_features = sales.iloc[:,3:].select_dtypes(include=["float64","int64" ])
plt.figure(figsize = (15, 10))
i = 1
for col in numerical_features:
    plt.subplot(2,3,i)
    sns.distplot(sales[col], hist=True, bins = 35)
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel("Count")
    i+=1
    plt.tight_layout()
```




```
# Check number of outliers
for feature in numerical_features:
    val = np.where(np.abs(stats.zscore(sales[feature])>3))
    print(f"{feature} column has {len(val[0])} outliers")
    print(val[0])
```

```
Impressions column has 23 outliers
[ 518  524  525  528  574  628  662  706  760  765  768  807  860  865
  867  884  909  912  995 1026 1123 1127 1138]
Clicks column has 33 outliers
[ 525  574  662  706  760  765  807  860  865  867  884  903  909  929
  937  949  969  970  995 1002 1003 1009 1025 1026 1027 1032 1035 1041
 1118 1123 1127 1134 1138]
Spent column has 32 outliers
[ 525  528  574  662  706  760  765  768  807  812  860  865  867  884
  903  909  929  949  969  970  995 1002 1003 1025 1026 1027 1032 1041
 1118 1123 1134 1138]
Total_Conversion column has 26 outliers
[ 518  524  525  528  531  544  561  568  574  577  579  613  628  706
  806  807  827  859  860  867 1094 1097 1101 1115 1116 1127]
Approved_Conversion column has 22 outliers
[ 518  524  525  528  531  544  561  574  577  579  613  662  765  806
  807  860  867 1032 1101 1115 1116 1127]
```

We can remove outliers from the next step

Check duplicate rows

```
sales[sales.duplicated()]
```

ad_id	xyz_campaign_id	fb_campaign_id	age	gender	interest	Impressions	Clicks	Spent	Total_Conversion	Approved_Conversion

Observation from the basic data understanding:

1. There 3 different campaigns that this company is using, and campaigns # 1178 has the most count, while campaign #916 has the least.
2. We have 4 different age groups, and age 30-34 has the most count here.

3. Age group 40-44 has most engagement in campaign #916, while age group 30-34 has the most engagement in campaign #1178. Campaign # 936 has relative the same engagement among all 4 age groups, and 30-34 is relatively higher.
4. Gender wise, more male than female in this dataset, and #916 & #936 male engagement is slightly higher than female group, while #1178 female has higher engagement.
5. Gender vs. Age: for group 1 (30-34) and group 2 (35-39): more male; while group 3 (40-44) and group 4 (45-49): more female
6. the more you spent on ads, the more impression of the ads and clicks it will get.

3. Data cleaning

1. categorical features
   - change xyz_campaign_id to categorical feature, and label it as 1,2,3
   - gender
   - age
2. outliers handling
3. features correlation matrix

```
# Categorical features
sales['age'] = sales['age'].replace(['30-34', '35-39','40-44','45-49'], [1,2,3,4])
sales['gender']=sales['gender'].replace(['M',"F"],[1,2])
sales['xyz_campaign_id']=sales['xyz_campaign_id'].replace([916,936,1178],[1,2,3])
sales.head()
```

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

```
# remove outliers with z_socre
def remove_outliers_z_score(dataframe, column, z = 3):
    dataframe["zscore"] = stats.zscore(dataframe[column])
    removed = dataframe[(dataframe["zscore"]<-z)|
                        (dataframe["zscore"]>z)].shape
    dataframe = dataframe[(dataframe["zscore"]>-z)&
                        (dataframe["zscore"]<z)]
    print(f"Removed: {removed[0]} outliers of {column}")
    return dataframe.drop(columns="zscore")
```

```
for feature in numerical_features:
    sales = remove_outliers_z_score(sales,feature)
```

```
Removed: 23 outliers of Impressions
Removed: 29 outliers of Clicks
Removed: 30 outliers of Spent
Removed: 22 outliers of Total_Conversion
Removed: 12 outliers of Approved_Conversion
```

```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1027 entries, 0 to 1142
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ad_id                1027 non-null   category
 1   xyz_campaign_id      1027 non-null   int64
 2   fb_campaign_id       1027 non-null   category
 3   age                  1027 non-null   int64
 4   gender               1027 non-null   int64
 5   interest             1027 non-null   category
 6   Impressions          1027 non-null   int64
 7   Clicks               1027 non-null   int64
 8   Spent                1027 non-null   float64
 9   Total_Conversion     1027 non-null   int64
 10  Approved_Conversion  1027 non-null   int64
dtypes: category(3), float64(1), int64(7)
memory usage: 153.1 KB
```

```
sales.skew()
# skewness level has dropped
```

```
xyz_campaign_id       -0.535637
age                    0.322063
gender                 0.134873
Impressions            1.909354
Clicks                 2.153875
Spent                  1.986033
Total_Conversion       2.304436
Approved_Conversion    1.162428
dtype: float64
```

From the correlation heatmap, we can tell that the impressions, clicks and spent are important features for approved conversion, now we will dig deeper by plotting the KPI

4. Plotting the KPIs

Our goal is to optimize sales conversion (Approved_Conversion) and predict future sales, first we need to understand some KPIs :

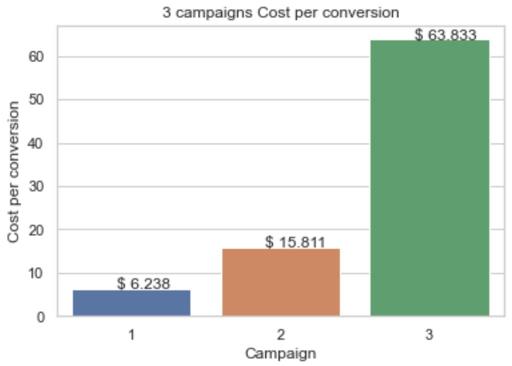
1. Return on ad spend (ROAS): but we don't have the revenue data here
2. Cost per conversion (CPA): is a great indicator of ROI, able to see which campaign is the most effective. For facebook ads CPA is defined as cost to Per New User Registration. CPA = Spent / approved_conversion
3. Click Through Rate （CTR） = Clicks/Impressions
4. Conversion rate （CvR） = Approved_conversion / Clicks
5. Cost per click （CPC） = Spent / Clicks

1. CPA

```
ads1 = sales[sales['xyz_campaign_id']==1]
ads2 = sales[sales['xyz_campaign_id']==2]
ads3 = sales[sales['xyz_campaign_id']==3]
# add CPA to the dataframe
sales['CPA'] = round(sales['Spent']/sales['Approved_Conversion'],3)

cc_1 = round(((ads1['Spent'].sum()/ads1['Approved_Conversion'].sum())),3)
cc_2 = round(((ads2['Spent'].sum()/ads2['Approved_Conversion'].sum())),3)
cc_3 = round(((ads3['Spent'].sum()/ads3['Approved_Conversion'].sum())),3)
cc_total = [cc_1, cc_2, cc_3]
x = [1, 2, 3]
print(cc_total)
plt.xlabel('Campaign')
plt.ylabel("Cost per conversion")
plt.title("3 campaigns Cost per conversion")
ax_cc = sns.barplot(x = x, y = cc_total)
# display horizon bar chart with value label
for p in ax_cc.patches:
    ax_cc.annotate('$ {:.3f}'.format(p.get_height()), (p.get_x()+0.3, p.get_height()))
```

```
[6.238, 15.811, 63.833]
```

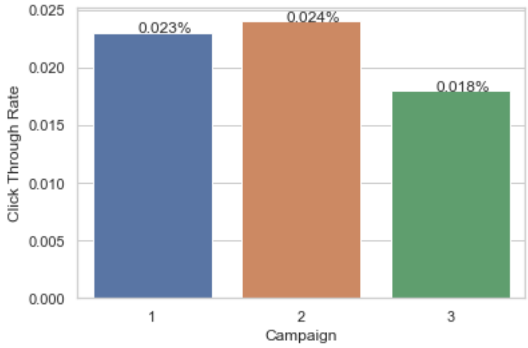


2. CTR

```
# add CTR to the dataframe
sales['CTR'] = round(sales['Clicks']*100/sales['Impressions'],3)
ctr_1 = round(((ads1['Clicks'].sum()/ads1['Impressions'].sum())*100),3)
ctr_2 = round(((ads2['Clicks'].sum()/ads2['Impressions'].sum())*100),3)
```

```
ctr_3 = round(((ads3['Clicks'].sum()/ads3['Impressions'].sum())*100),3)
ctr_total = [ctr_1, ctr_2, ctr_3]
print(ctr_total)
plt.xlabel('Campaign')
plt.ylabel("Click Through Rate")
ax_ctr = sns.barplot(x = x, y = ctr_total)
for p in ax_ctr.patches:
    ax_ctr.annotate('{:.3f}%'.format(p.get_height()), (p.get_x()+0.3, p.get_height()))
```

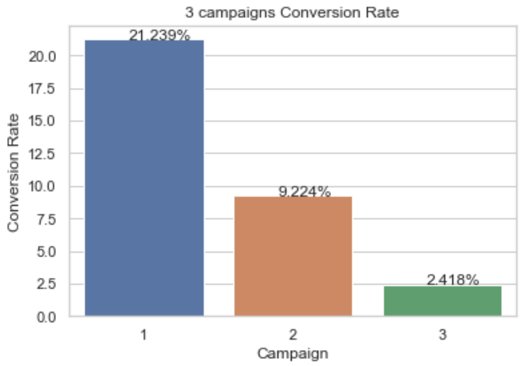
```
[0.023, 0.024, 0.018]
```



3. CvR (Conversion Rate)

```
# add CTR to the dataframe
sales['CvR'] = round(sales['Approved_Conversion']*100/sales['Clicks'],3)
con_1 = round(((ads1['Approved_Conversion'].sum()/ads1['Clicks'].sum())*100),3)
con_2 = round(((ads2['Approved_Conversion'].sum()/ads2['Clicks'].sum())*100),3)
con_3 = round(((ads3['Approved_Conversion'].sum()/ads3['Clicks'].sum())*100),3)
con_total = [con_1, con_2, con_3]
print(con_total)
plt.xlabel('Campaign')
plt.ylabel("Conversion Rate")
plt.title("3 campaigns Conversion Rate")
ax_cr = sns.barplot(x = x, y = con_total)
# display horizon bar chart with value label
for p in ax_cr.patches:
    ax_cr.annotate('{:.3f}%'.format(p.get_height()), (p.get_x()+0.3, p.get_height()))
```

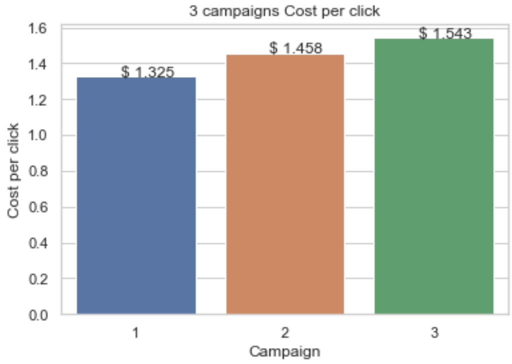
```
[21.239, 9.224, 2.418]
```



4. CPC

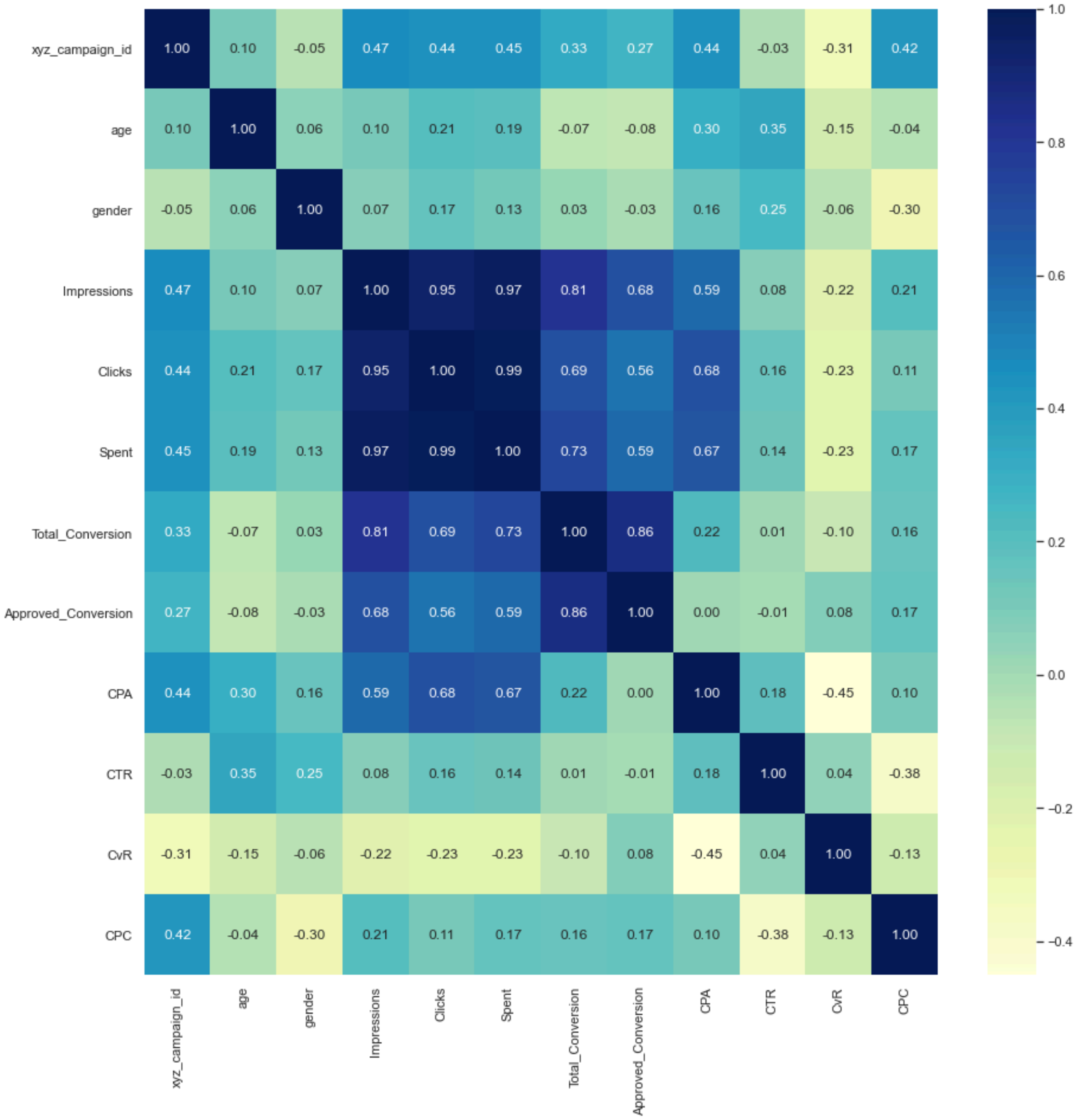
```
# add CTR to the dataframe
sales['CPC'] = round(sales['Spent']/sales['Clicks'],3)
cpc_1 = round(((ads1['Spent'].sum()/ads1['Clicks'].sum())),3)
cpc_2 = round(((ads2['Spent'].sum()/ads2['Clicks'].sum())),3)
cpc_3 = round(((ads3['Spent'].sum()/ads3['Clicks'].sum())),3)
cpc_total = [cpc_1, cpc_2, cpc_3]
print(cpc_total)
plt.xlabel('Campaign')
plt.ylabel("Cost per click")
plt.title("3 campaigns Cost per click")
ax_cpc = sns.barplot(x = x, y = cpc_total)
# display horizon bar chart with value label
for p in ax_cpc.patches:
    ax_cpc.annotate('$ {:.3f}'.format(p.get_height()), (p.get_x()+0.3, p.get_height()))
```

```
[1.325, 1.458, 1.543]
```




```
f,ax = plt.subplots(figsize=(15,15))
sns.heatmap(sales.corr(),annot = True, fmt=".2f", cmap = "YlGnBu")
```

```
<AxesSubplot:>
```



Observations from KPIs

1. Campaign 1 has the lowest cost per conversion and cost per click, and the second highest click through rate, with the highest conversion rate. In sum, campaign 1 is the most efficient campaign with lowest cost.
2. Campaign 3 has the highest cost per conversion and cost per click, which has the highest cost, but the click through rate and conversion rate is the lowest. We need to check the campaign design to find out why users not likely to click the ads, and why the content is not motivating user to make the purchase.

3. Campaign 2 has the highest CTR, which means this campaign design is attracting user to click the ads, but the conversion rate is relatively low. We need to dig deeper and see after clicking the ads, what does it leads to? And why is not motivating user to make the purchase.

```
sales = sales.dropna()
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 936 entries, 0 to 1142
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ad_id                936 non-null    category
 1   xyz_campaign_id      936 non-null    int64
 2   fb_campaign_id       936 non-null    category
 3   age                  936 non-null    int64
 4   gender               936 non-null    int64
 5   interest             936 non-null    category
 6   Impressions          936 non-null    int64
 7   Clicks               936 non-null    int64
 8   Spent                936 non-null    float64
 9   Total_Conversion     936 non-null    int64
 10  Approved_Conversion  936 non-null    int64
 11  CPA                  936 non-null    float64
 12  CTR                  936 non-null    float64
 13  CvR                  936 non-null    float64
 14  CPC                  936 non-null    float64
dtypes: category(3), float64(5), int64(7)
memory usage: 175.5 KB
```

5. Modeling

Define x and y

```
x = sales.drop(labels=["Approved_Conversion", "Total_Conversion","ad_id","xyz_campaign_id","fb_campaign_id","CPA", "CTR","CvR","CPC"],a:
y = sales["Total_Conversion"]
x.columns
```

```
Index(['age', 'gender', 'interest', 'Impressions', 'Clicks', 'Spent'], dtype='object')
```

```
scaler = StandardScaler()
x = scaler.fit_transform(x)
```

```
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size = 0.25, random_state = 42)
```

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

```
LinearRegression()
```

```
import math
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    mae = mean_absolute_error(test_labels, predictions)
    mse = mean_squared_error(test_labels, predictions)
#     errors = abs(predictions - test_labels)
    rmse = np.sqrt(mean_squared_error(test_labels, predictions))
#     mape = 100 * np.mean(errors / test_labels)
#     accuracy = 100-mape

    print('Model Performance')
    print('Mean Absolute Error: {:0.4f} degrees.'.format(mae))
    print('Mean Squared Error: {:0.4f} degrees.'.format(mae))
    print('Root Mean Squared Error: {:0.4f} degrees.'.format(rmse))
    print('R2 = {:0.5f} %.'.format(model.score(test_features, test_labels)*100))
#     print('Accuracy =: {:0.4f}%.'.format(accuracy))
#     return accuracy
```

```
evaluate(model, X_test, y_test)
```

```
Model Performance
Mean Absolute Error: 1.2766 degrees.
```

```
Mean Squared Error: 1.2766 degrees.
Root Mean Squared Error: 2.6366 degrees.
R2 = 67.28263 %.
```

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 200, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
random_grid
```

```
{'n_estimators': [10, 31, 52, 73, 94, 115, 136, 157, 178, 200],
 'max_features': ['auto', 'sqrt'],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'min_samples_split': [2, 5, 10],
 'min_samples_leaf': [1, 2, 4],
 'bootstrap': [True, False]}
```

```
rf = RandomForestRegressor()
rf_random = rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_st
rf_random.fit(X_train, y_train)
rf_random.best_params_
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    2.5s
[Parallel(n_jobs=-1)]: Done 130 tasks      | elapsed:    4.1s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:    6.8s finished
{'n_estimators': 73,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
 'max_features': 'auto',
 'max_depth': None,
 'bootstrap': True}
```

```
base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(X_train, y_train)
base_accuracy = evaluate(base_model, X_test, y_test)
```

```
Model Performance
Mean Absolute Error: 1.2342 degrees.
Mean Squared Error: 1.2342 degrees.
Root Mean Squared Error: 2.3313 degrees.
R2 = 74.42035 %.
```

```
best_random = rf_random.best_estimator_
random_accuracy = evaluate(best_random, X_test, y_test)
```

```
Model Performance
Mean Absolute Error: 1.1302 degrees.
Mean Squared Error: 1.1302 degrees.
Root Mean Squared Error: 2.2521 degrees.
R2 = 76.13042 %.
```

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'bootstrap': bootstrap,
    'max_depth': max_depth,
    'max_features': max_features,
    'min_samples_leaf': min_samples_leaf,
    'min_samples_split': min_samples_split,
    'n_estimators':  n_estimators
}
rf = RandomForestRegressor()
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)
```

```
grid_search.fit(X_train, y_train)
grid_search.best_params_
```

```
Fitting 3 folds for each of 4320 candidates, totalling 12960 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 228 tasks      | elapsed:    3.6s
[Parallel(n_jobs=-1)]: Done 634 tasks      | elapsed:    9.3s
[Parallel(n_jobs=-1)]: Done 1200 tasks      | elapsed:   17.8s
[Parallel(n_jobs=-1)]: Done 1930 tasks      | elapsed:   29.1s
[Parallel(n_jobs=-1)]: Done 2820 tasks      | elapsed:   42.8s
[Parallel(n_jobs=-1)]: Done 3874 tasks      | elapsed:   59.1s
[Parallel(n_jobs=-1)]: Done 5088 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 6466 tasks      | elapsed:  1.7min
[Parallel(n_jobs=-1)]: Done 8004 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done 9706 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 11568 tasks      | elapsed:  3.0min
[Parallel(n_jobs=-1)]: Done 12960 out of 12960 | elapsed:  3.4min finished
{'bootstrap': True,
 'max_depth': 10,
 'max_features': 'auto',
 'min_samples_leaf': 4,
 'min_samples_split': 5,
 'n_estimators': 10}
```

```
best_grid = grid_search.best_estimator_
grid_accuracy = evaluate(best_grid, X_test, y_test)
```

```
Model Performance
Mean Absolute Error: 1.1729 degrees.
Mean Squared Error: 1.1729 degrees.
Root Mean Squared Error: 2.4650 degrees.
R2 = 71.40319 %.
```

Define x and y

```
x = sales.drop(labels=["Approved_Conversion", "Total_Conversion","ad_id","xyz_campaign_id","fb_campaign_id","age","gender","CPA", "CTR"
y = sales["Total_Conversion"]
x.columns
```

```
Index(['interest', 'Impressions', 'Clicks', 'Spent'], dtype='object')
```

Scaling

```
scaler = StandardScaler()
x = scaler.fit_transform(x)
```

Splitting Data

```
X_train, X_test, y_train, y_test = train_test_split(x,y,test_size = 0.25, random_state = 42)
```

Linear Regression Model

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

```
LinearRegression()
```

```
import math
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    mae = mean_absolute_error(test_labels, predictions)
    mse = mean_squared_error(test_labels, predictions)
#     errors = abs(predictions - test_labels)
    rmse = np.sqrt(mean_squared_error(test_labels, predictions))
#     mape = 100 * np.mean(errors / test_labels)
#     accuracy = 100-mape

    print('Model Performance')
    print('Mean Absolute Error: {:0.4f} degrees.'.format(mae))
    print('Mean Squared Error: {:0.4f} degrees.'.format(mae))
```

```
    print('Root Mean Squared Error: {:0.4f} degrees.'.format(rmse))
    print('R2 = {:0.5f} %.'.format(model.score(test_features, test_labels)*100))
#     print('Accuracy =: {:0.4f}%.'.format(accuracy))
#     return accuracy
```

```
evaluate(model, X_test, y_test)
```

```
Model Performance
Mean Absolute Error: 1.2608 degrees.
Mean Squared Error: 1.2608 degrees.
Root Mean Squared Error: 2.6615 degrees.
R2 = 66.66164 %.
```

Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(n_estimators = 10, random_state = 42)
rfr.fit(X_train, y_train)
y_predict = rfr.predict(X_test)
y_predict = np.round(y_predict) # user conversion should be integer
print(y_predict[:15])
print(list(y_test[:15]))
```

```
[ 1.  1.  1. 29.  1.  1.  2.  1.  4.  2.  2.  1.  1.  2.  4.]
[1, 1, 1, 38, 1, 1, 5, 1, 3, 1, 1, 1, 1, 2, 6]
```

```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
mae=mean_absolute_error(y_test, y_predict)
mse=mean_squared_error(y_test, y_predict)
rmse=np.sqrt(mse)
r2_score=r2_score(y_test, y_predict)
```

```
mae,r2_score
```

```
(1.2735042735042734, 0.6942880128720836)
```

Random Hyperparameter Grid for Random Forest Regressor

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 200, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
random_grid
```

```
{'n_estimators': [10, 31, 52, 73, 94, 115, 136, 157, 178, 200],
 'max_features': ['auto', 'sqrt'],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'min_samples_split': [2, 5, 10],
 'min_samples_leaf': [1, 2, 4],
 'bootstrap': [True, False]}
```

```
rf = RandomForestRegressor()
rf_random = rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_st
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    2.4s
[Parallel(n_jobs=-1)]: Done 130 tasks      | elapsed:    4.1s
```

```
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:    6.5s finished
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [10, 31, 52, 73, 94,
                                                         115, 136, 157, 178,
                                                         200]},
                   random_state=42, verbose=2)
```

```
rf_random.best_params_
```

```
{'n_estimators': 73,
 'min_samples_split': 5,
 'min_samples_leaf': 2,
```