PREDICT CTR (Click Through Rate) Of EMAILS

The goal of this project is to maximize the Click Through Rate (CTR) of emails for clients. Email communication is one of the popular ways for companies pitch products to their users and build trustworthy relationships with them.

Problem Statement:

To build a machine learning-based approach to predict the CTR of an email campaign.

Information on Dataset:

The data holds information of past email campaigns containing the email attributes like subject and body length, no. of CTA, date and time of an email, type of the audience, whether its a personalized email or not, etc and the target variable indicating the CTR of the email campaign.

Data Dictionary:

Data for the project is taken from kaggle and contains 3 files - train.csv, test.csv and data_to_test.csv

Train and Test Set:

Train and Test set contains different sets of email campaigns containing information about the email campaign. Train set includes the target variable click_rate and you need to predict the click_rate of an email campaign in the test set.

Dataset Variable Description:

campaign_id - Unique identifier of a campaign

sender - Sender of an e-mail

subject_len - No. of characters in a subject

body_len - No. of characters in an email body

mean_paragraph_len - Average no. of characters in paragraph of an email

day_of_week - Day on which email is sent

is_weekend - Boolean flag indicating if an email is sent on weekend or not times_of_day - Times of day when email is sent: Morning, Noon, Evening

category - Category of the product an email is related to

product - Type of the product an email is related to

no_of_CTA - No. of Call To Actions in an email

mean_CTA_len - Average no. of characters in a CTA

is_image - No. of images in an email

is_personalised - Boolean flag indicating if an email is personalized to the user or not

is_quote - No. of quotes in an email

is_timer - Boolean flag indicating if an email contains a timer or not

is_emoticons - No. of emoticons in an email

is_discount - Boolean flag indicating if an email contains a discount or not

is_price - Boolean flag indicating if an email contains price or not

is_urgency - Boolean flag indicating if an email contains urgency or not

target_audience - Cluster label of the target audience

click_rate (Target Variable) - Click rate of an email campaign

.

MODEL TESTING

Evaluation File Format:

data_to_test.csv contains 2 variables - campaign id and click_rate

Variable - Description:

campaign_id - Unique Identifier of a campaign id click_rate (Target Variable) - Click rate of an email campaign

Evaluation metric:

The evaluation metric for this project would be r2_score.

Data Split:

Test data is further divided into Public (40%) and Private (60%) data. Public data is generic data that attract customers based on interests and offers. Private data is sourced from companies targetting customers for particular product. This split between the two, trains the model for world scenarios and can be later enhanced using advance ML models.

1. Import the required libraries

In [ ]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

2. Data Inspection

In [ ]:
```
# Read the datasets

train = pd.read_csv('../input/jobathon-august-2022/train_F3fUq2S.csv')
test = pd.read_csv('../input/jobathon-august-2022/test_Bk2wfZ3.csv')
```

In [ ]:
```
# check the shapes of the dataset

print('No.of rows and columns in train dataset',train.shape, '\n')
print('No.of rows and columns in test dataset',test.shape)
```

```
No.of rows and columns in train dataset (1888, 22)

No.of rows and columns in test dataset (762, 21)
```

We have 1888 rows and 22 columns in Train set whereas Test set has 762 rows and 21 columns.

In [ ]:
```
# Read the first 5 rows of train and test datasets

train.head()
```

Out[ ]:

	campaign_id	sender	subject_len	body_len	mean_paragraph_len	day_of_week	is_weekend
0	1	3	76	10439	39	5	1
1	2	3	54	2570	256	5	1
2	3	3	59	12801	16	5	1
3	4	3	74	11037	30	4	0
4	5	3	80	10011	27	5	1

5 rows × 22 columns

In [ ]:
```
test.head()
```

Out[ ]:

	campaign_id	sender	subject_len	body_len	mean_paragraph_len	day_of_week	is_weekend
0	1889	3	61	12871	11	6	1
1	1890	3	54	2569	256	5	1
2	1891	3	88	1473	78	4	0
3	1892	3	88	1473	78	3	0
4	1893	3	78	9020	29	3	0

5 rows × 21 columns

In [ ]:
```
# Info about the train and test datsets

train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1888 entries, 0 to 1887
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   campaign_id         1888 non-null   int64  
 1   sender              1888 non-null   int64  
 2   subject_len         1888 non-null   int64  
 3   body_len            1888 non-null   int64  
 4   mean_paragraph_len  1888 non-null   int64  
 5   day_of_week         1888 non-null   int64  
 6   is_weekend          1888 non-null   int64  
 7   times_of_day        1888 non-null   object 
 8   category            1888 non-null   int64  
 9   product             1888 non-null   int64  
 10  no_of_CTA           1888 non-null   int64  
 11  mean_CTA_len        1888 non-null   int64  
 12  is_image            1888 non-null   int64  
 13  is_personalised     1888 non-null   int64  
 14  is_quote            1888 non-null   int64  
 15  is_timer            1888 non-null   int64  
 16  is_emoticons        1888 non-null   int64  
 17  is_discount         1888 non-null   int64  
 18  is_price            1888 non-null   int64  
 19  is_urgency          1888 non-null   int64  
 20  target_audience     1888 non-null   int64  
 21  click_rate          1888 non-null   float64
dtypes: float64(1), int64(20), object(1)
memory usage: 324.6+ KB
```

In the train dataset, we have 1 categorical feature and 20 numerical features

In [ ]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 762 entries, 0 to 761
Data columns (total 21 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   campaign_id         762 non-null    int64 
 1   sender              762 non-null    int64 
 2   subject_len         762 non-null    int64 
 3   body_len            762 non-null    int64 
 4   mean_paragraph_len  762 non-null    int64 
 5   day_of_week         762 non-null    int64 
 6   is_weekend          762 non-null    int64 
 7   times_of_day        762 non-null    object
 8   category            762 non-null    int64 
 9   product             762 non-null    int64 
 10  no_of_CTA           762 non-null    int64 
 11  mean_CTA_len        762 non-null    int64 
 12  is_image            762 non-null    int64 
 13  is_personalised     762 non-null    int64 
 14  is_quote            762 non-null    int64 
 15  is_timer            762 non-null    int64 
 16  is_emoticons        762 non-null    int64 
 17  is_discount         762 non-null    int64 
 18  is_price            762 non-null    int64 
 19  is_urgency          762 non-null    int64 
 20  target_audience     762 non-null    int64 
dtypes: int64(20), object(1)
memory usage: 125.1+ KB
```

In the test dataset, we have 1 categorical feature and 19 numerical features

3. Data Cleaning

- **Check for Missing values**

Before we go on to build the model, we must look for missing values within the dataset as treating the missing values is a necessary step before we fit a machine learning model on the dataset.

In [ ]:
```
train.isnull().sum()
```

Out[ ]:
```
campaign_id           0
sender                0
subject_len           0
body_len              0
mean_paragraph_len    0
day_of_week           0
is_weekend            0
times_of_day          0
category              0
product               0
no_of_CTA             0
mean_CTA_len          0
is_image              0
is_personalised       0
is_quote              0
is_timer              0
is_emoticons          0
is_discount           0
is_price              0
is_urgency            0
target_audience       0
click_rate            0
dtype: int64
```

In [ ]:
```
test.isnull().sum()
```

Out[ ]:
```
campaign_id            0
sender                 0
subject_len            0
body_len               0
mean_paragraph_len     0
day_of_week            0
is_weekend             0
times_of_day           0
category               0
product                0
no_of_CTA              0
mean_CTA_len           0
is_image               0
is_personalised        0
is_quote               0
is_timer               0
is_emoticons           0
is_discount            0
is_price               0
is_urgency             0
target_audience        0
dtype: int64
```

In the train and test datasets, we don't have any null values.

4. Exploratary Data Analysis (EDA)

In [ ]:
```
# Features in train and test datasets

train.columns
```

Out[ ]:
```
Index(['campaign_id', 'sender', 'subject_len', 'body_len',
       'mean_paragraph_len', 'day_of_week', 'is_weekend', 'times_of_
day',
       'category', 'product', 'no_of_CTA', 'mean_CTA_len', 'is_imag
e',
       'is_personalised', 'is_quote', 'is_timer', 'is_emoticons',
       'is_discount', 'is_price', 'is_urgency', 'target_audience',
       'click_rate'],
      dtype='object')
```

In [ ]:
```
test.columns
```

Out[ ]:
```
Index(['campaign_id', 'sender', 'subject_len', 'body_len',
       'mean_paragraph_len', 'day_of_week', 'is_weekend', 'times_of_
day',
       'category', 'product', 'no_of_CTA', 'mean_CTA_len', 'is_imag
e',
       'is_personalised', 'is_quote', 'is_timer', 'is_emoticons',
       'is_discount', 'is_price', 'is_urgency', 'target_audience'],
      dtype='object')
```

- **Target Variable**

In this section we will take a look at the 'click_rate' (CTR) of an email campaign which is the target variable. It is crucial to understand it in detail as this is what we are trying to predict accurately.

In [ ]:
```
train['click_rate'].describe()
```

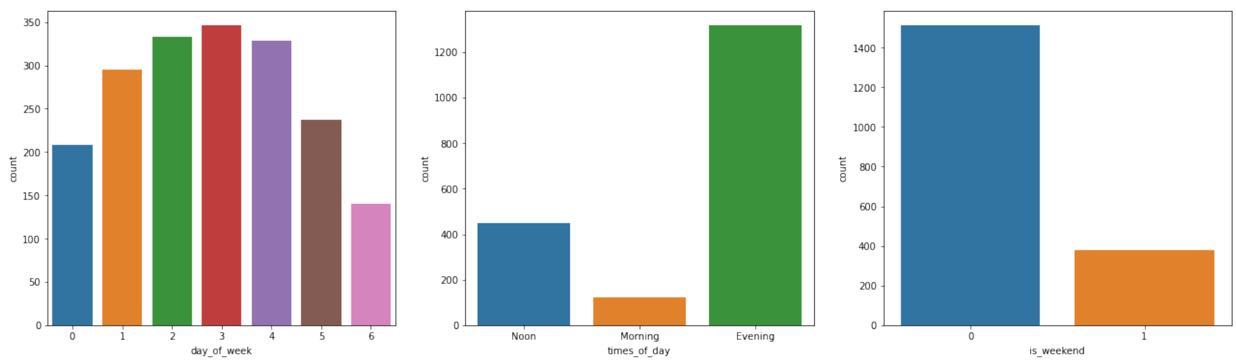
Out[ ]:
```
count    1888.000000
mean        0.041888
std         0.084223
min         0.000000
25%         0.005413
50%         0.010686
75%         0.035589
max         0.897959
Name: click_rate, dtype: float64
```

The target variable (Click Through Rate) has a max of 89% Click rate.

- **Univariate Analysis**

In [ ]:
```
# Binary Features

plt.figure(figsize=(22,6))

# Day of week
plt.subplot(1,3,1)
sns.countplot('day_of_week',data=train)

# Times of day
plt.subplot(1,3,2)
sns.countplot('times_of_day',data=train)

# Weekend or not
plt.subplot(1,3,3)
sns.countplot('is_weekend',data=train)
```

Out[ ]:
```
<AxesSubplot:xlabel='is_weekend', ylabel='count'>
```



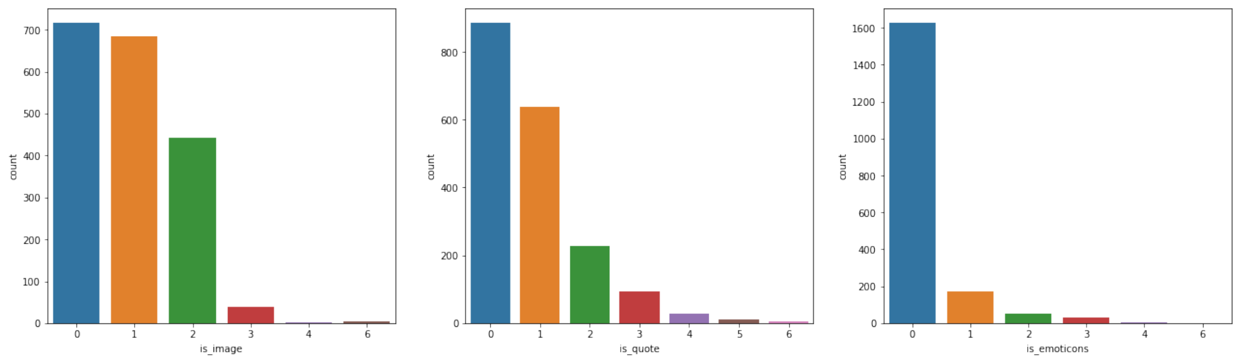
Assume that 0-6 indicates Sunday to Saturday, as most of the emails were sent on wednesday, tuesday and thrusaday.

Most of the emails were sent during evenings as people were free during most of that time.

Assume that 0 --> Not Weekend, 1 --> Weekend, as most of the emails were sent on weekdays and less no.of emails were sent on weekends.

In [ ]:
```
plt.figure(figsize=(22,6))

# No.of Images in an email
plt.subplot(1,3,1)
sns.countplot('is_image',data=train)

# No.of quotes in an email
plt.subplot(1,3,2)
sns.countplot('is_quote',data=train)

# No. of emoticons in an email
plt.subplot(1,3,3)
sns.countplot('is_emoticons',data=train)
```

Out[ ]:
```
<AxesSubplot:xlabel='is_emoticons', ylabel='count'>
```



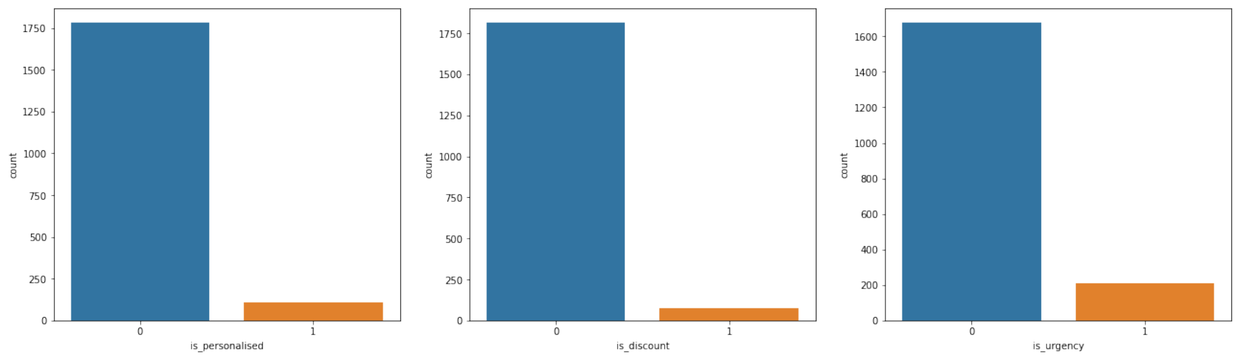
Assume that 0 to 6 indicates no.of images in an email. Since email containing 0-2 images are more and with 3-6 images are less

Email containing 0-1 quotes are more and with 2-6 are less

Email containing 0 emotions/emojis are more and with 1-6 are less

In [ ]:
```
plt.figure(figsize=(22,6))

# Personalized emails or not
plt.subplot(1,3,1)
sns.countplot('is_personalised',data=train)

# Discount email or not
plt.subplot(1,3,2)
sns.countplot('is_discount',data=train)

# Urgent email or not
plt.subplot(1,3,3)
sns.countplot('is_urgency',data=train)
```

Out[ ]: <AxesSubplot:xlabel='is_urgency', ylabel='count'>



Most the emails were not personalized which can be special discounts, offers emails and less no.of emails were personalized which can be related towards their work.

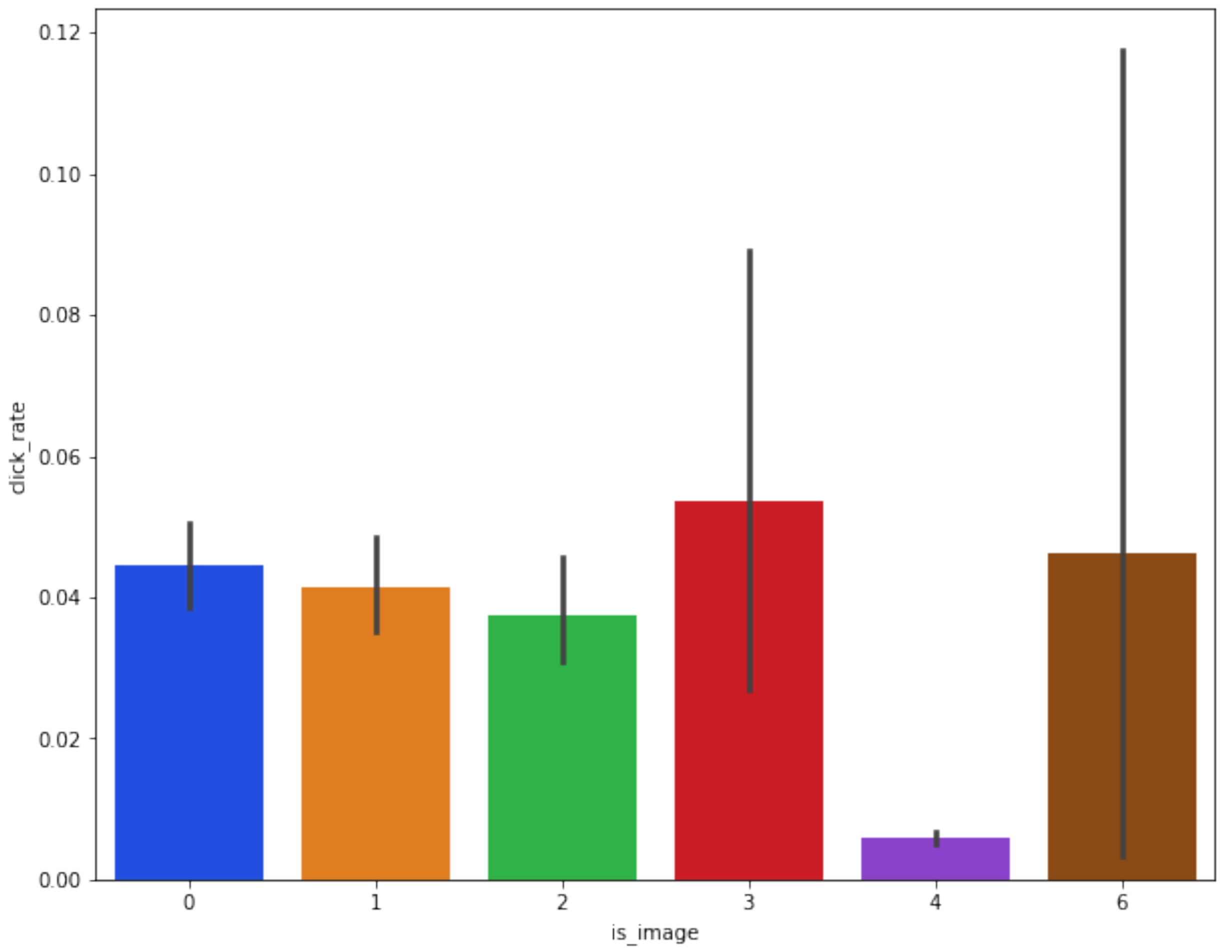
Most of them were not discount emails and less no.of emails were discounted emails since most of the discount emails get during sales period.

Most of the emails were not important/urgency emails and less no.of emails were urgency emails since it can be related towards their work.

- **Bivariate Analysis**

In [ ]:
```
# Click rate vs Image

plt.figure(figsize=(10,8))
sns.barplot(x='is_image',y='click_rate',data=train,palette='bright')
```

Out[ ]: <AxesSubplot:xlabel='is_image', ylabel='click_rate'>

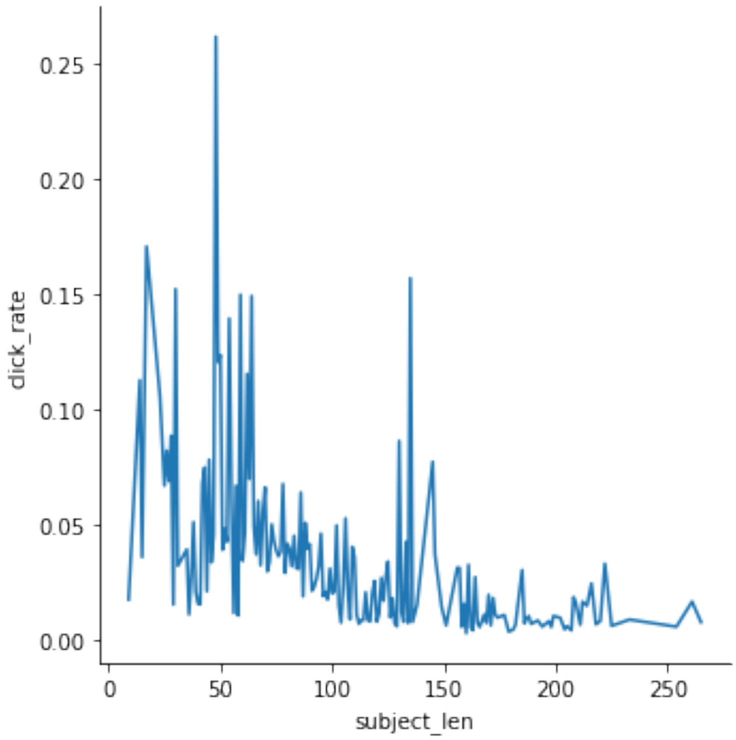


0-2 images in an email are more but click rate for 6 and 3 images in an email seems higher and hence CTR can be maxmized by providing more images in an email.

In [ ]:
```
# click rate vs subject length

plt.figure(figsize=(20,8))
sns.relplot(x="subject_len", y="click_rate",ci=None,kind="line", data=
train)
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7fbe8dbe2d10>

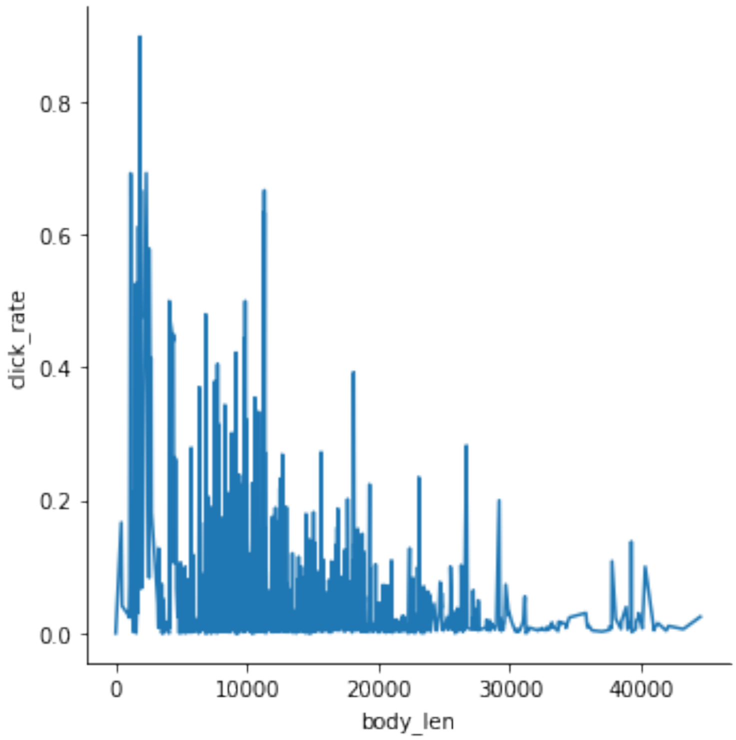
<Figure size 1440x576 with 0 Axes>



If the no.of characters in a subject of an email is 50 then CTR can be maximized.

In [ ]:
```
# click rate vs length of an email

plt.figure(figsize=(20,8))
sns.relplot(x="body_len", y="click_rate",ci=None,kind="line", data=tra
in)
```

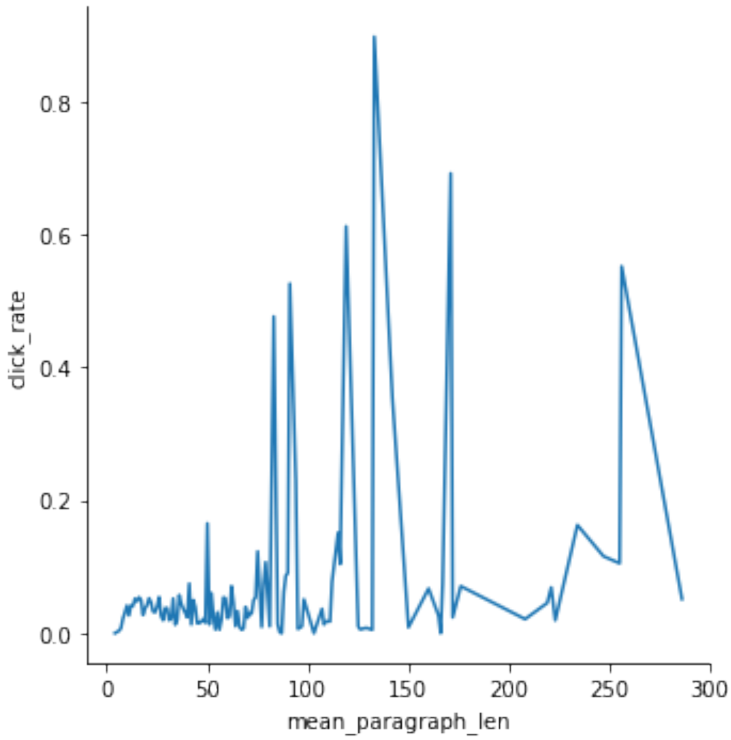
Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7fbe8dba27d0>

<Figure size 1440x576 with 0 Axes>
```



If the No. of characters in an email body is in the range of 100-200 then CTR can be maxmized.

In [ ]:
```
# click rate vs Mean paragraph length of an email

plt.figure(figsize=(20,8))
sns.relplot(x="mean_paragraph_len", y="click_rate",ci=None,kind="lin
e", data=train)
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x7fbe8dbb52d0>

<Figure size 1440x576 with 0 Axes>
```



To maxmize the CTR the Average no. of characters in paragraph of an email should be in the range of 130-150.

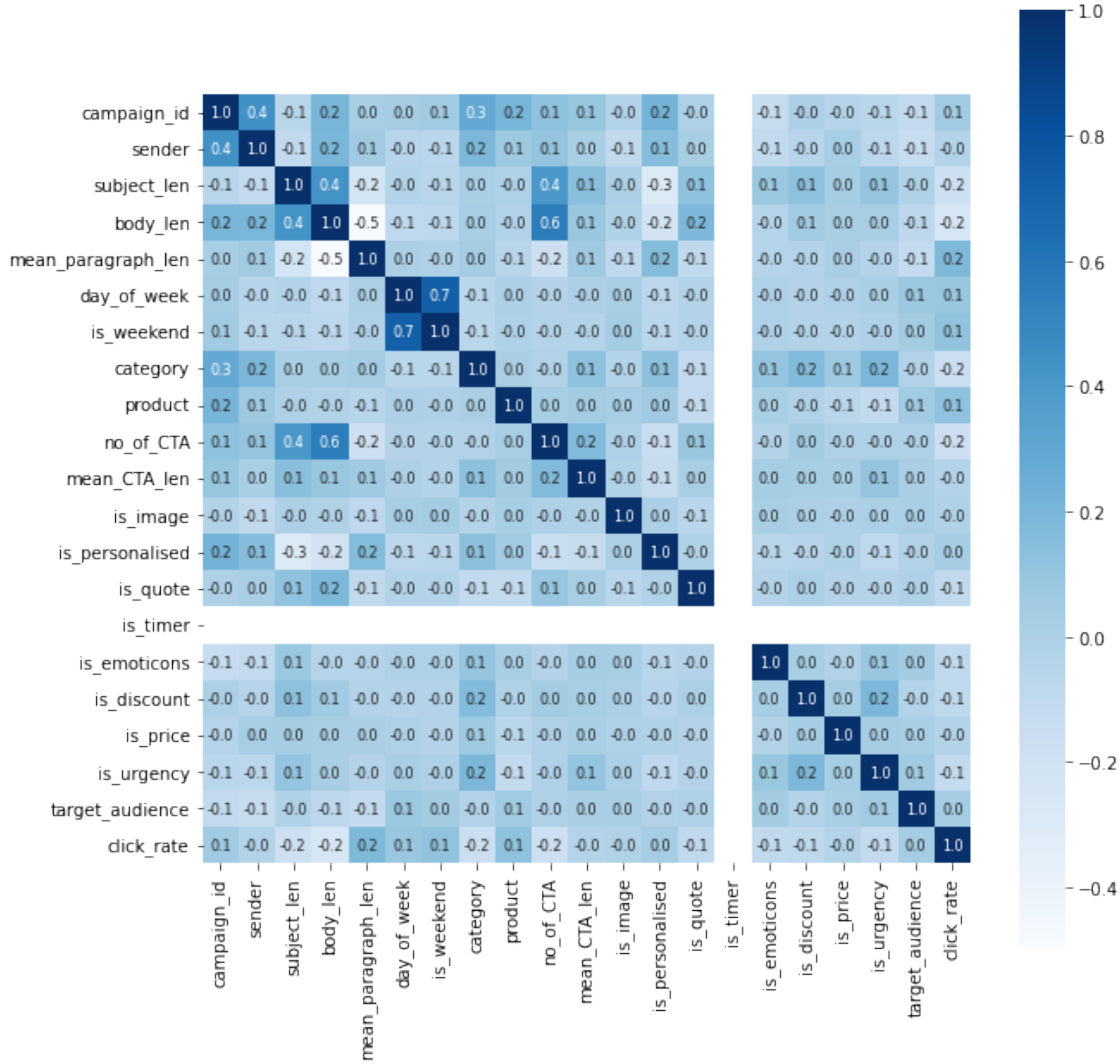
- **Correlation Heat Map**

Understanding the correlation between various features in the dataset

In [ ]:
```
correlation = train.corr()
```

In [ ]:
```
# constructing a heatmap to understand the correlation

plt.figure(figsize=(10,10))
sns.heatmap(correlation, cbar=True, square=True, fmt='.1f', annot=True, annot_kws={'size':8}, cmap='Blues')
```

Out[ ]: <AxesSubplot:>



5. Data Pre-Processing

- **Label Encoding to the Categorical features**

Here only 'times of day' is the only categorical feature

In [ ]:
```
print(train['times_of_day'].value_counts(),'\n')
print(test['times_of_day'].value_counts(),'\n')
```

```
Evening    1317
Noon        447
Morning     124
Name: times_of_day, dtype: int64

Evening    532
Noon       175
Morning     55
Name: times_of_day, dtype: int64
```

In [ ]:
```
# Import Label encoder from sklearn

from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
# Define the model
le = LabelEncoder()

var_mod = train.select_dtypes(include='object').columns
for i in var_mod:
    train[i] = le.fit_transform(train[i])

for i in var_mod:
    test[i] = le.fit_transform(test[i])
```

In [ ]:
```
train.head()
```

Out[ ]:

	campaign_id	sender	subject_len	body_len	mean_paragraph_len	day_of_week	is_weekend
0	1	3	76	10439	39	5	1
1	2	3	54	2570	256	5	1
2	3	3	59	12801	16	5	1
3	4	3	74	11037	30	4	0
4	5	3	80	10011	27	5	1

5 rows × 22 columns

In [ ]:
```
test.head()
```

Out[ ]:

	campaign_id	sender	subject_len	body_len	mean_paragraph_len	day_of_week	is_weekend
0	1889	3	61	12871	11	6	1
1	1890	3	54	2569	256	5	1
2	1891	3	88	1473	78	4	0
3	1892	3	88	1473	78	3	0
4	1893	3	78	9020	29	3	0

5 rows × 21 columns

The labels in the 'times of day' feature has changed to numerical data in the train and test data.

__Here 1--> Morning, 2--> Noon, 0--> Evening

6. Model Building

In [ ]:
```
# Import train test split from sklearn

from sklearn.model_selection import train_test_split
```

In [ ]:
```
# Splitting the data into Features and Traget

X = train.drop(['click_rate'],axis=1)
Y = train['click_rate']
```

In [ ]:
```
print(X, '\n')
print(Y)
```

```
      campaign_id  sender  subject_len  body_len  mean_paragraph_len
\
0               1       3           76     10439                  39
1               2       3           54      2570                 256
2               3       3           59     12801                  16
3               4       3           74     11037                  30
4               5       3           80     10011                  27
...           ...     ...          ...       ...                 ...
1883         1884       3           88      1451                  75
1884         1885       3           58     10537                  40
1885         1886       3           89     11050                  26
```

```
1886         1887      3          58    10537                40
1887         1888      3          89    11050                26

      day_of_week  is_weekend  times_of_day  category  product  ...
\
0               5           1             2         6       26  ...
1               5           1             1         2       11  ...
2               5           1             2         2       11  ...
3               4           0             0        15        9  ...
4               5           1             2         6       26  ...
...           ...         ...           ...       ...      ...  ...
1883            2           0             2         2       11  ...
1884            2           0             0         2       11  ...
1885            1           0             0        15        9  ...
1886            1           0             0         2       11  ...
1887            0           0             0        15        9  ...

      mean_CTA_len  is_image  is_personalised  is_quote  is_timer  \
0               29         0                0         0         0
1               22         0                0         0         0
2               23         1                0         1         0
3               24         0                0         0         0
4               31         0                0         1         0
...            ...       ...              ...       ...       ...
1883            22         0                0         1         0
1884            27         0                0         0         0
1885            28         0                0         0         0
1886            27         0                0         0         0
1887            28         0                0         0         0

      is_emoticons  is_discount  is_price  is_urgency  target_audien
ce
0                0            0         0           0
14
1                0            0         0           0
10
2                0            0         0           0
16
3                0            0         0           0
10
4                0            0         0           0
14
...            ...          ...       ...         ...
...
1883             0            0         0           0
10
1884             0            0         0           0
11
1885             0            0         0           0
6
```

```
1886              0            0          0            0
16
1887              0            0          0            0
10

[1888 rows x 21 columns]

0       0.103079
1       0.700000
2       0.002769
3       0.010868
4       0.142826
          ...
1883    0.350746
1884    0.004728
1885    0.008289
1886    0.012014
1887    0.003644
Name: click_rate, Length: 1888, dtype: float64
```

In [ ]:
```python
# Splitting the data into Training data and Test data(20%)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
0.2, random_state = 22)
```

In [ ]:
```python
print(X.shape, X_train.shape, X_test.shape)
```

```
(1888, 21) (1510, 21) (378, 21)
```

7. Development with ML Models

In [ ]:
```python
# Import the ML models libraries

from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from xgboost import XGBRegressor
from sklearn import metrics
```

In [ ]:
```
algos = [LinearRegression(), Lasso(), Ridge(), KNeighborsRegressor(),
DecisionTreeRegressor(), XGBRegressor()]

names = ['Linear Regression', 'Lasso Regression', 'Ridge Regression',
'K Neighbors Regressor', 'Decision Tree Regressor', 'XGBoost Regresso
r']

r2_score_list = []
```

In [ ]:
```
for name in algos:
    model = name                                    # Load the model
    model.fit(X_train, Y_train)                     # Fit the model with traini
ng data
    test_data_pred = model.predict(X_test)          # prediction on test
data(i.e Y_pred)
    r2 = metrics.r2_score(Y_test, test_data_pred)   # R2 error
    r2_score_list.append(r2)
```

In [ ]:
```
evaluation = pd.DataFrame({'Model': names, 'r2': r2_score_list})
```

In [ ]:
```
evaluation
```

Out[ ]:

	Model	r2
0	Linear Regression	0.121362
1	Lasso Regression	0.063618
2	Ridge Regression	0.121473
3	K Neighbors Regressor	0.272447
4	Decision Tree Regressor	0.096715
5	XGBoost Regressor	0.557550

8. Conclusion and Submission

As we can clearly see XGBoost Regressor performs slighlty better than KNeighbours Regressor, Linear, Ridge and Lasso regression and Decision Tree Regressor do not improve the score so we can select XGBoost Regressor for making our final predictions.

Make a Submission to CSV file

In [ ]:
```
submission = pd.read_csv('../input/jobathon-august-2022/sample_submiss
ion_LJ2N3ZQ.csv')
model = XGBRegressor()
model.fit(X, Y)
final_predictions = model.predict(test)
submission['click_rate'] = final_predictions
```

In [ ]:
```
print(final_predictions)
```

```
[ 1.29891619e-01  1.57037526e-01  2.78526187e-01  2.78526187e-01
  1.06823526e-01  8.34494606e-02  7.21955625e-03  3.16429022e-03
  4.23383638e-02  3.04214731e-02  2.40662601e-02  8.71517658e-02
  2.16489262e-03  1.91694945e-02  4.83820774e-02  3.83185223e-02
  2.47685723e-02  1.63237359e-02  2.51350459e-02  4.19223271e-02
  1.31289652e-02  1.81701202e-02  1.88425332e-02  2.87642926e-02
  5.21754585e-02  4.05700468e-02  1.17324367e-01  9.06611420e-03
  1.21842369e-01  4.45922390e-02  5.31758033e-02  1.53314874e-01
 -2.91830394e-03  1.72261506e-01  5.48477322e-02  4.74582799e-02
  8.00144486e-03  3.06841265e-03  3.90467630e-03  6.99584857e-02
  2.98358011e-03  7.53475577e-02 -1.75360870e-03  8.87777284e-02
  9.19376388e-02  6.06419984e-03  1.54249053e-02  3.24616842e-02
  5.73674850e-02  6.59291213e-03  7.78059363e-02  1.84330102e-02
  1.97515134e-02  4.78561148e-02  9.46500991e-03  4.17413898e-02
  1.12136908e-01 -2.83422647e-03  1.33699924e-02  1.81023311e-02
  1.05095536e-01  2.84031499e-02  3.16898674e-02  1.78819173e-03
  3.01263537e-02  1.80937815e-02  5.52594708e-03  1.31780043e-01
  6.54961355e-03  7.21370205e-02  1.13997040e-02  2.70824432e-01
  8.16840213e-03  4.05232534e-02  1.33113399e-01  2.03532457e-01
  3.41570638e-02  1.25845283e-01  1.43427858e-02  1.37383863e-01
 -1.59024708e-02  7.08093215e-03  1.39335915e-01  1.74008477e-02
 -8.53322260e-03  2.30008841e-01  1.97353549e-02  3.16993669e-02
  1.31062105e-01  1.39518473e-02  6.04330860e-02  3.43829468e-02
  7.59122148e-02  4.29171249e-02  1.30732497e-02  6.23001046e-02
  5.67973778e-02  1.31278068e-01  1.83562413e-02  3.38438489e-02
  1.47760376e-01  1.47760376e-01  1.47760376e-01  4.45788121e-03
  4.67064828e-02  8.45287368e-03  4.68532741e-02  5.34862950e-02
  7.24423751e-02  4.49153967e-02  8.45287070e-02  5.74738868e-02
  1.06652029e-01  8.58137012e-02 -1.51175847e-02  4.44791000e-03
  2.09320650e-01  1.04054101e-01  5.72474115e-03  5.81740700e-02
  4.66592163e-02  8.28935429e-02  1.25030845e-01 -1.88624405e-03
  1.19400034e-02  3.93867977e-02  2.33975006e-04  1.16207348e-02
 -7.73677602e-03  1.98577694e-03  5.55548519e-02  9.44674909e-02
  6.68653054e-03  5.99118136e-02  7.78741986e-02  8.80326107e-02
  2.79605761e-03  6.16136231e-02  5.42698763e-02  1.27037046e-02
  1.53172365e-03  1.08335108e-01  1.86716393e-02  1.22732282e-01
  1.23896047e-01 -2.47811642e-03  7.43406191e-02  1.24867834e-01
  3.94214801e-02  8.11920911e-02  1.21629797e-01  9.25503597e-02
  2.30293974e-01  3.52812861e-03  7.94836134e-02  5.99718234e-03
  1.34083675e-03  5.98914595e-03  1.76582765e-02  5.50783612e-03
```

```
  4.41376343e-02  2.55574614e-01 -1.74921949e-03  2.31152475e-02
  9.94165242e-03  1.28144221e-02  2.02234648e-02  2.05951244e-01
  3.56061989e-03 -9.43677593e-03  8.83768499e-02  2.70465594e-02
  7.36030862e-02  1.66561767e-01  5.73433749e-03  1.43699184e-01
  2.10589617e-02  1.71715748e-02  1.46373257e-01 -2.36100494e-03
  7.31988177e-02  2.01536030e-01  1.20459544e-02  2.74694040e-02
  1.58604863e-03  3.61288674e-02 -4.77670552e-03  4.49540764e-02
  2.17441991e-01  6.34169504e-02  7.75279179e-02  6.55155629e-02
  4.58616577e-02  9.40051824e-02  1.13082431e-01  1.59195393e-01
  6.20062761e-02  4.50497167e-03  8.81959721e-02  5.83055429e-02
  8.16833824e-02  7.75487870e-02  6.72318786e-02  3.16745276e-03
  1.52897224e-01  5.08385561e-02  9.11741555e-02  9.27586481e-02
  1.99633613e-01  3.47464234e-02  1.53608933e-01  2.89199036e-02
  1.71455555e-02  8.24950039e-02  2.86446605e-02  1.67283252e-01
  1.56043172e-01  8.08460489e-02  1.86835807e-02  4.23174314e-02
  7.93330520e-02  2.06986040e-01  1.02637641e-01  6.97278157e-02
 -9.46445949e-03  2.11739317e-02  1.05883643e-01  9.80395898e-02
  9.07379575e-03  6.77543581e-02  5.44716464e-03  1.25951627e-02
  9.89860520e-02  4.34410200e-02 -3.70733556e-03  6.88871602e-03
  9.93580092e-03  1.05307341e-01  5.63438376e-03  1.67817762e-03
 -6.44996855e-03  9.30950865e-02 -5.78456849e-04  1.05277179e-02
  3.05208806e-02  4.70538111e-03  5.84694138e-03  1.93981186e-03
  5.32196797e-02  1.09394472e-02  7.81264827e-02  3.13018262e-02
  3.20770293e-02  9.70536396e-02  3.75425480e-02  1.81933027e-02
  4.06449661e-02  8.21571518e-03 -4.47920640e-04  2.14022752e-02
  1.06565049e-02  3.81209217e-02  1.98881060e-01  2.37744637e-02
  2.68794689e-02 -1.47762452e-03  1.60736144e-01  3.12023070e-02
  4.75855777e-03  2.95410510e-02  1.69317834e-02  7.33284280e-02
  2.71833688e-02  1.91384152e-01 -5.47795603e-03  3.49604040e-02
  7.64983371e-02  6.97719455e-02  5.29191978e-02  6.51708618e-02
  6.54093623e-02  2.90750768e-02  3.29013243e-02  5.05999960e-02
  3.47222053e-02  6.36795089e-02  3.64740714e-02  6.36795089e-02
  1.10658877e-01  3.13877314e-02  4.67286520e-02  4.94888723e-02
  5.96308261e-02  1.17644481e-02  1.48743987e-01  1.10261850e-01
  4.40640785e-02  8.74490943e-03  7.63330087e-02  1.61456034e-01
  2.74991319e-02  8.78721997e-02  1.38902411e-01  8.11861530e-02
  1.19622014e-02  2.34142076e-02  1.89228468e-02  1.30018920e-01
  1.89570203e-01 -4.51041013e-03  3.59343961e-02  3.56004797e-02
  1.06404582e-02  8.97307694e-02  2.29819082e-02  2.44525727e-02
  2.08314389e-01  1.35829374e-01  1.55565171e-02  4.44330415e-03
  4.27637585e-02  5.85696241e-03  7.08645359e-02  2.64622606e-02
 -4.71552194e-04  4.09471951e-02  6.24332810e-03 -6.09382614e-03
  1.16808370e-01 -1.51235587e-03  6.03160858e-02  2.16947086e-02
  8.52001458e-03 -7.89895467e-03  3.27763222e-02  7.44230002e-02
  5.15076555e-02  9.06584188e-02  5.80941699e-02  2.23790435e-03
  6.55842423e-02 -1.27137813e-03  2.47045904e-01  1.63278937e-01
  3.18486214e-01  1.04948163e-01  5.47827594e-02  4.47648019e-03
  9.96121438e-04 -2.65897193e-04  3.59955370e-01  3.27128917e-02
  1.58248339e-02  7.05741532e-03  2.42399368e-02 -1.21180806e-02
  4.54668254e-02 -3.62820458e-03  1.07058641e-02  3.79619934e-02
```

```
  5.87182641e-02  3.72099057e-02  2.83993641e-03  1.91850448e-03
  1.90339033e-02 -2.89431890e-03 -2.89431890e-03  4.72290851e-02
  2.74673179e-02 -8.57327948e-04  9.52636972e-02 -5.46982838e-03
 -9.35538672e-04  1.03412062e-01 -6.32704794e-03  1.73105597e-02
  3.64534892e-02  3.89127731e-02  1.39109893e-02  1.95433617e-01
  7.95860775e-03  1.65476277e-02  7.57932803e-03  1.70965627e-01
  1.41404748e-01  6.79821800e-03 -3.03871022e-03  1.43847913e-01
  1.39621422e-01  9.42088570e-03  1.22542530e-01  7.94879273e-02
 -1.07646547e-02  1.59491063e-03  9.17057246e-02  1.42471895e-01
  5.06593427e-03  7.10720941e-02  3.99778858e-02 -1.07275806e-02
  1.08049490e-01  9.98551492e-03  9.54878107e-02 -1.19788032e-02
  1.87231286e-03  4.71777208e-02 -1.16861993e-02  5.39902225e-02
 -2.30764714e-03  2.82805711e-02  1.09443768e-04  5.32556958e-02
  1.21000014e-01  1.95613317e-02 -2.71937833e-03  1.85666718e-02
  1.12672895e-02  3.81613821e-02  7.75158405e-02  4.42380160e-02
  2.15443230e-04  2.13503987e-02  1.34684956e-02  1.15269743e-01
  2.71539087e-03  2.96530258e-02  2.12423354e-02  2.10730843e-02
  7.41800293e-02  8.70692059e-02  4.75877598e-02  1.45269886e-01
  3.57573628e-02  9.25389305e-03  4.11822870e-02  7.99238458e-02
  9.65288840e-03  4.19962928e-02  2.14481242e-02  1.39548227e-01
  1.91196185e-02 -1.56532973e-03 -7.91765377e-03  1.09818712e-01
  4.56548035e-02  5.20968735e-02  9.42650661e-02  1.60479080e-02
  6.14883043e-02  9.20780972e-02  2.85110716e-03  2.63123624e-02
  7.15201069e-03  2.94528641e-02  6.82591740e-03  1.02448184e-02
  1.16009735e-01  5.36962569e-01  8.78510103e-02  6.34076595e-02
  6.93553388e-02  8.68233759e-03  8.19091201e-02  3.96940559e-02
  5.32706916e-01  7.14789033e-02  1.75179522e-02  9.44315642e-02
  6.36235029e-02  9.14281234e-02  1.88681316e-02  5.69034182e-02
  1.07525639e-01  7.43854642e-02  2.79819649e-02  4.15147748e-03
  2.97904275e-02  2.51153186e-02 -9.51693859e-03  1.52172476e-01
  2.57269472e-01  3.80459167e-02  1.13170043e-01  8.26858431e-02
  4.45286781e-02  1.38647944e-01  6.05906807e-02  9.69214737e-02
  1.39414251e-01  5.99005520e-02  3.77731398e-02  3.43023129e-02
  1.32791027e-01  9.49190110e-02  2.85324790e-02  7.33131021e-02
  7.58778583e-03  3.07965204e-02  1.33898705e-01  8.98730829e-02
  1.48844630e-01  8.84966180e-02  1.41947582e-01  6.87240958e-02
  1.39499471e-01  1.44772336e-01  2.61609294e-02  1.16954148e-02
  5.04208952e-02  3.12693059e-01  4.08961438e-02  4.74010482e-02
  4.57742102e-02  6.63758963e-02  9.11006257e-02  5.96027486e-02
  6.78972751e-02  2.81820986e-02  5.71490899e-02  6.69347718e-02
  6.58400878e-02  3.80200520e-02  2.19434798e-01  9.90366042e-02
 -3.88901718e-02  4.42221820e-01  3.03577725e-02  3.61474603e-02
  4.82577197e-02  1.17521487e-01  4.75154109e-02  7.30925351e-02
  7.84240738e-02  1.67053223e-01  8.91855434e-02  6.18876405e-02
  1.81226414e-02  1.33526968e-02  1.43391946e-02  3.49561721e-02
  5.29796332e-02  5.63464798e-02  2.20077131e-02  1.08614177e-01
  5.54390885e-02  1.83544457e-01 -2.64081871e-03  1.50158539e-01
  7.55261406e-02  2.44500175e-01  1.60806086e-02  1.59750298e-01
  4.53020111e-02  1.33239895e-01  5.54458573e-02  5.62973730e-02
  7.65250996e-02  1.32834002e-01  6.02014065e-02  9.53679904e-02
```

```
 8.51424634e-02  1.83736142e-02  2.11551972e-02  4.06825952e-02
 1.19085284e-02  5.46925850e-02  5.30487001e-02  3.47696692e-02
 5.78491911e-02  4.20464575e-02  2.05359384e-02  1.06545143e-01
 6.99001085e-03  8.55139866e-02  1.33621737e-01  9.29750595e-03
 6.53025508e-02  1.27825784e-02  2.08518598e-02  1.24544054e-01
 7.91783184e-02  1.68834720e-02  7.88534284e-02  8.35983306e-02
 7.86035061e-02  5.66376150e-02  3.95344235e-02  1.73677094e-02
 3.17223668e-02 -8.64744093e-03  1.40951248e-02  2.00303039e-03
 6.58128932e-02  5.53216320e-03  1.00493565e-01  3.28878462e-02
 2.85226442e-02  1.08118974e-01  4.20464575e-02  9.47274938e-02
 7.98674077e-02  6.38929680e-02  6.97947219e-02 -2.66547687e-03
 1.03867233e-01  1.86554529e-02  6.70485897e-03  6.76767575e-03
 7.34145567e-02  1.26770243e-01 -1.13559305e-03  5.05456850e-02
 4.74165827e-02  5.62863313e-02  1.63019359e-01  1.63399745e-02
 9.00353119e-02  1.06558032e-01  1.18418485e-02  1.88315231e-02
 4.70291227e-02  2.55486779e-02  2.61010118e-02  2.51013357e-02
 3.75069268e-02  7.05781430e-02  7.04931617e-02  6.53205439e-03
 1.57509521e-01  9.21222046e-02  2.46711507e-01  1.68180898e-01
-8.98593757e-03  2.99074538e-02  1.03170179e-01  5.83636314e-02
 8.45280588e-02  5.06571382e-02  7.09876418e-02  4.47582826e-02
 1.17920958e-01  1.82670742e-01  1.15096398e-01  7.22204968e-02
 3.88674662e-02  3.61448117e-02  8.65340009e-02  2.42742617e-02
 4.69347313e-02  1.40871271e-01  2.59354077e-02  2.29556989e-02
 8.93765092e-02  1.35121942e-02  7.59427389e-03  3.71338464e-02
 3.84533629e-02  7.09947804e-03  6.36545941e-02  1.63788542e-01
 2.05884963e-01  2.57959962e-02  9.35737044e-02  1.73270181e-02
 6.40051365e-02  3.54648978e-02  9.89724845e-02  4.83152419e-02
 1.05060004e-01  6.17052652e-02  2.81598140e-02  4.84567732e-02
 7.56537542e-02  1.32043839e-01  4.35811952e-02  2.87743751e-02
 2.64613517e-02  3.88184488e-02  1.60787478e-01  1.53620124e-01
 1.16321065e-01  9.68942717e-02  1.97864044e-02  4.46996577e-02
 2.48022061e-02  6.41677454e-02  4.66740578e-02  5.38026541e-02
 5.93665801e-02  3.25747058e-02  1.57653093e-02  1.98820457e-02
 1.02994122e-01  9.74346176e-02  9.55234095e-02  3.38605680e-02
 6.41457140e-02  6.55042753e-02  9.44485422e-03  4.74168956e-02
 1.38833880e-01  5.38375825e-02  4.28922959e-02  6.53593689e-02
 1.65694043e-01  1.07271485e-01  2.86937784e-02  7.51625821e-02
 8.33171010e-02  3.27506736e-02  2.08746970e-01  2.26419587e-02
 4.38386276e-02  2.64711659e-02  3.23675424e-02  4.58596162e-02
 7.57214054e-02  8.63779932e-02  4.36073840e-02  6.70417920e-02
 1.86644614e-01  1.83939651e-01  2.45797075e-02  1.35326073e-01
 3.62336710e-02  7.36936033e-02  8.99444222e-02  1.14542320e-01
 4.04741615e-02  1.08164586e-01  2.30923548e-01  1.09722212e-01
 7.82202110e-02  7.61584342e-02  3.61322574e-02  1.74194090e-02
 1.38764689e-02  2.28940379e-02  4.64573205e-02  7.10056424e-02
 7.98915699e-02  5.72908744e-02  7.50398859e-02  4.54732552e-02
 1.34622883e-02  4.03023586e-02  1.29227996e-01  8.18597078e-02
 4.59546782e-02  8.51911306e-02  1.96117088e-01  1.62566617e-01
 3.60298693e-01  7.34415976e-03  3.26608390e-01  1.66507233e-02
 2.84659952e-01  5.18686138e-04  1.34189716e-02  5.58408734e-04
```

```
 3.45168680e-01 -2.87445635e-03]
```

In [ ]:
```
#only positive predictions for the target variable

#submission['click_rate'] = submission['click_rate'].apply(lambda x: 0 if x<0 else x)
submission.to_csv('my_submission.csv', index=False)
```

In [ ]: