In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 150)

from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

In [2]:

```
trainData = pd.read_csv('train.csv')
testData = pd.read_csv('test.csv')
```

In [3]:

```
#Shape of datasets
print(trainData.shape)
print(testData.shape)
```

```
(4209, 378)
(4209, 377)
```

In [4]:

```
#Categorical vs numerical columns
print(trainData.dtypes.value_counts(), end='\n\n')
print(testData.dtypes.value_counts())
#in test dataset target column is not there(float)
```

```
int64      369
object       8
float64      1
dtype: int64

int64     369
object      8
dtype: int64
```

In [5]:

```
#Columns with null values

print('Null Values in Train Dataset: ', trainData.isnull().sum().where(lambda x : x>0).drop
print('Null Values in Test Dataset: ', testData.isnull().sum().where(lambda x : x>0).dropna
```

```
Null Values in Train Dataset:  Series([], dtype: float64)
Null Values in Test Dataset:  Series([], dtype: float64)
```

In [6]:

```
#Identify Categorical Columns
train_cat_features = trainData.select_dtypes(include='object').columns
test_cat_features = testData.select_dtypes(include='object').columns
print('Categorical Features in Train Dataset : ', train_cat_features)
print('Categorical Features in Test Dataset : ', test_cat_features)
```

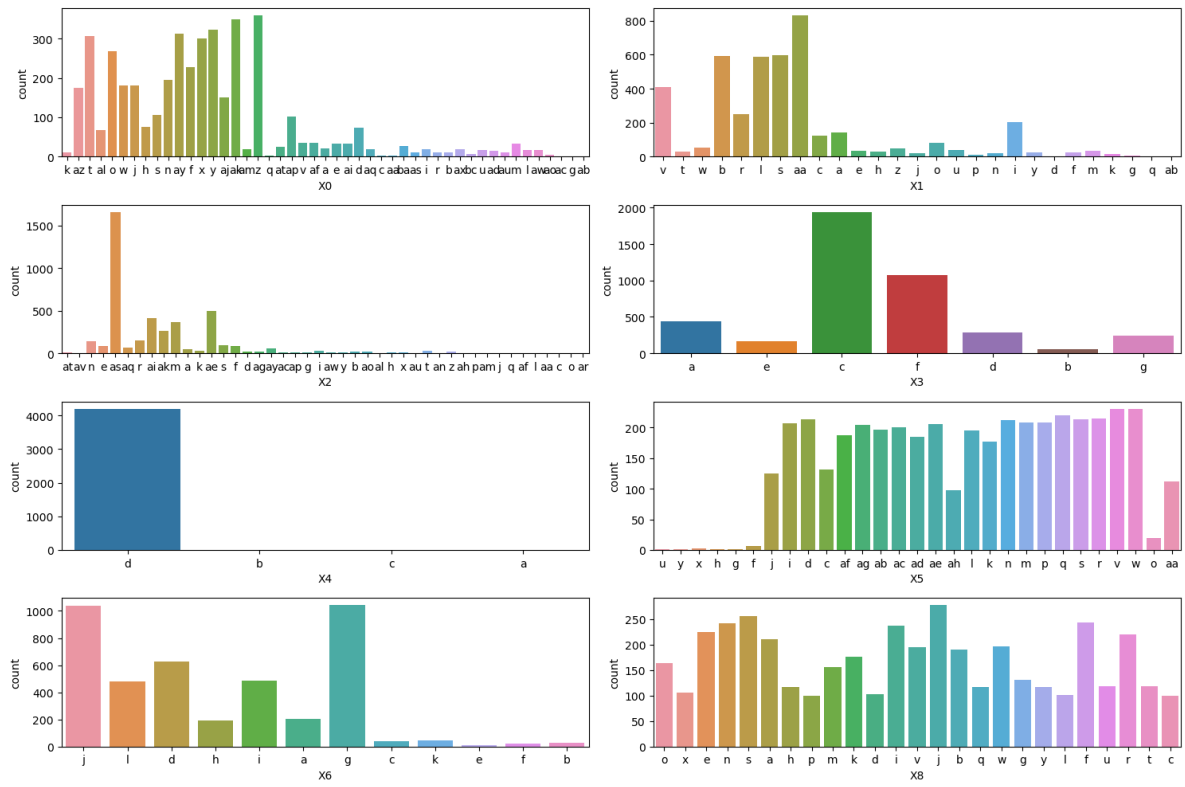
```
Categorical Features in Train Dataset :  Index(['X0', 'X1', 'X2', 'X3', 'X
4', 'X5', 'X6', 'X8'], dtype='object')
Categorical Features in Test Dataset :  Index(['X0', 'X1', 'X2', 'X3', 'X4',
'X5', 'X6', 'X8'], dtype='object')
```

In [7]:

```
#Visualize Distribution of EMan time vs these catagorical columns to find out any implied i
fig = plt.figure(figsize=(15, 10))

for index, col in enumerate(train_cat_features):
    plt.subplot(4, 2, index+1)
    sns.countplot(x=col,data=trainData[train_cat_features])

plt.tight_layout()
plt.show();
```

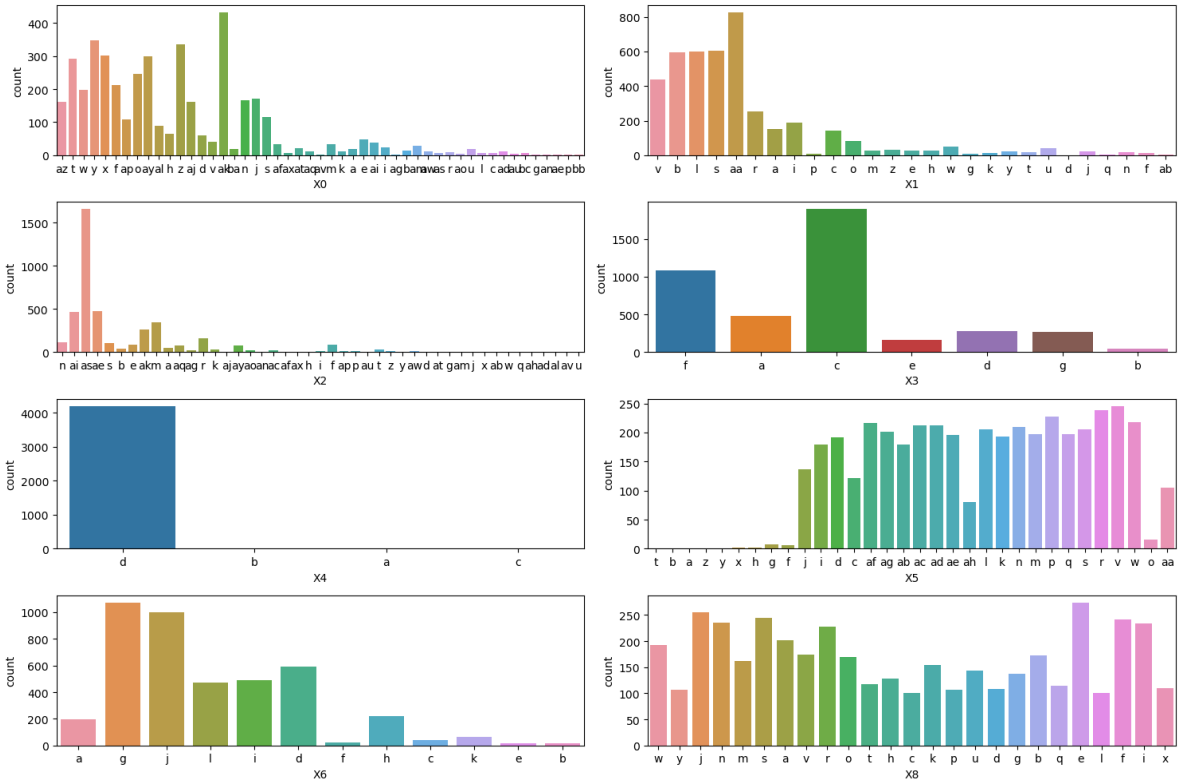


In [8]:

```
fig = plt.figure(figsize=(15, 10))

for index, col in enumerate(test_cat_features):
    plt.subplot(4, 2, index+1)
    sns.countplot(x=col,data=testData[train_cat_features])

plt.tight_layout()
plt.show();
```



In [9]:

```
cat_features = set(train_cat_features) - set(['X4'])
trainData.drop(columns=['X4','ID'], inplace=True)
testData.drop(columns=['X4','ID'], inplace=True)
```

In [10]:

```
counts1 = [[], []]
for c in trainData.columns:
    if c not in cat_features:
        typ = trainData[c].dtype
        uniq = len(np.unique(trainData[c]))
        if uniq == 1: counts1[0].append(c)
        elif uniq == 2 and typ == np.int64: counts1[1].append(c)

print('Constant features: {} \nBinary features: {} \n'.format(*[len(c) for c in counts1]))

print('Constant features:', counts1[0])
```

```
Constant features: 12
Binary features: 356

Constant features: ['X11', 'X93', 'X107', 'X233', 'X235', 'X268', 'X289', 'X
290', 'X293', 'X297', 'X330', 'X347']
```

In [11]:

```
counts2 = [[], []]
for c in testData.columns:
    if c not in cat_features:
        typ = testData[c].dtype
        uniq = len(np.unique(testData[c]))
        if uniq == 1: counts2[0].append(c)
        elif uniq == 2 and typ == np.int64: counts2[1].append(c)

print('Constant features: {} \nBinary features: {} \n'.format(*[len(c) for c in counts2]))

print('Constant features:', counts2[0])
```

```
Constant features: 5
Binary features: 363

Constant features: ['X257', 'X258', 'X295', 'X296', 'X369']
```

In [13]:

```
#Zero Variance Columns(Constant Values)
print('Constant features in train dataset')
for cname in counts1[0]:
    print(trainData[cname].unique(), end = ' ')

print('Constant features in test dataset')
for cname in counts2[0]:
    print(testData[cname].unique(), end=' ')
```

```
Constant features in train dataset
[0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] [0] Constant features in test da
taset
[0] [0] [0] [0] [0]
```

In [14]:

```
#Drop Constant Features(Deleting the same features from both training and testing data)
trainData.drop(columns=counts1[0], inplace=True)
testData.drop(columns=counts1[0], inplace=True)
```

In [15]:

```
print(trainData.shape)
print(testData.shape)
```

```
(4209, 364)
(4209, 363)
```

In [16]:

```
#Label Encode 1st 8 categrocal variables

for i in cat_features:
    enc = LabelEncoder()
    trainData[i]=enc.fit_transform(trainData[i])
    testData[i]=enc.fit_transform(testData[i])
```

In [17]:

```
trainX = trainData.drop(columns=['y'])
trainY = trainData['y']
```

In [18]:

```
pca =PCA(n_components=0.99, random_state=102)
trainXReduced =pca.fit_transform(trainX)
print(trainX.shape)
print(trainXReduced.shape)

testXReduced =pca.transform(testData)
print(testData.shape)
print(testXReduced.shape)
```

```
(4209, 363)
(4209, 27)
(4209, 363)
(4209, 27)
```

In [19]:

```
xtrain, xtest, ytrain, ytest = train_test_split(trainXReduced, trainY, test_size=0.25, rand
```

In [20]:

```
#Simple Model
xgb=XGBRegressor()
xgb.fit(xtrain,ytrain)
```

Out[20]:

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=Non
e,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

In [21]:

```
ypred = xgb.predict(xtest)
print('MSE: ', mean_squared_error(ytest, ypred))
print('RMSE: ', np.sqrt(mean_squared_error(ytest, ypred)))
print('MAE: ', mean_absolute_error(ytest, ypred))
#The normal case is when the R2 score is between zero and one like 0.8 which means your mod
print('R Squared: ', r2_score(ytest, ypred))
```

```
MSE:  116.41953965137267
RMSE:  10.789788675009936
MAE:  6.545406922198203
R Squared:  0.3976253648060962
```

In [22]:

```
#Hyper Parameter Tunnig
parameters = {
    'learning_rate': [0.05, 0.01, 0.1],
    'n_estimators': [500, 1000],
    'max_depth': [5, 8]
}


gsc = GridSearchCV(estimator=xgb, param_grid=parameters, cv=KFold(n_splits=3), scoring='neg
grid_result = gsc.fit(trainXReduced, trainY)
print('Best Params: ', grid_result.best_params_)
```

```
Best Params:  {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 500}
```

In [23]:

```
gscr = pd.DataFrame(grid_result.cv_results_)
for i in range(gscr.shape[0]):
    print(gscr.loc[i,'params'], gscr.loc[i,'mean_test_score'])
```

```
{'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 500} -89.99668185274
629
{'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 1000} -94.1637931551
6173
{'learning_rate': 0.05, 'max_depth': 8, 'n_estimators': 500} -89.55068307109
64
{'learning_rate': 0.05, 'max_depth': 8, 'n_estimators': 1000} -90.9459235753
261
{'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 500} -81.94577270550
437
{'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 1000} -83.9903798877
3662
{'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 500} -82.84236194945
231
{'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 1000} -85.7766943499
1458
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 500} -95.558803938813
01
{'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 1000} -97.71292433608
225
{'learning_rate': 0.1, 'max_depth': 8, 'n_estimators': 500} -91.020276132646
06
{'learning_rate': 0.1, 'max_depth': 8, 'n_estimators': 1000} -91.36112184680
995
```

In [24]:

```
xgb2=XGBRegressor(learning_rate=0.01, max_depth=6, n_estimators=500)
xgb2.fit(xtrain,ytrain)
ypred2 = xgb2.predict(xtest)
print('MSE: ', mean_squared_error(ytest, ypred2))
```

```
MSE:  106.58284501778452
```

In [25]:

```
xgb3=XGBRegressor(learning_rate=0.05, max_depth=6, n_estimators=1000)
xgb3.fit(xtrain,ytrain)
ypred3 = xgb3.predict(xtest)
print('MSE: ', mean_squared_error(ytest, ypred3))
```

```
MSE:  114.97797760208236
```

In [26]:

```
#Hence we can use params {'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 500} to pr
xgb2.predict(testXReduced)
```

Out[26]:

```
array([ 78.8135 ,  92.99192,  78.43336, ...,  93.58581, 104.76562,
        97.45147], dtype=float32)
```

In [ ]: