### Loading Required Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn import metrics

!pip install category_encoders
import category_encoders as ce

from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline



# Plotting options
mpl.style.use('ggplot')
sns.set(style='whitegrid')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Futu
reWarning: pandas.util.testing is deprecated. Use the functions in the public
API at pandas.testing instead.
  import pandas.util.testing as tm

Requirement already satisfied: category_encoders in /usr/local/lib/python3.6/
dist-packages (2.2.2)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python
3.6/dist-packages (from category_encoders) (0.22.2.post1)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.6/dist
-packages (from category_encoders) (1.18.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.6/dist-
packages (from category_encoders) (0.5.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.
6/dist-packages (from category_encoders) (0.10.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-
packages (from category_encoders) (1.4.1)
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.6/dis
t-packages (from category_encoders) (1.0.3)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-
packages (from scikit-learn>=0.20.0->category_encoders) (0.14.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from patsy>=0.5.1->category_encoders) (1.12.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-
packages (from pandas>=0.21.1->category_encoders) (2018.9)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/pytho
n3.6/dist-packages (from pandas>=0.21.1->category_encoders) (2.8.1)
```

### Mount Google Drive

In [2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

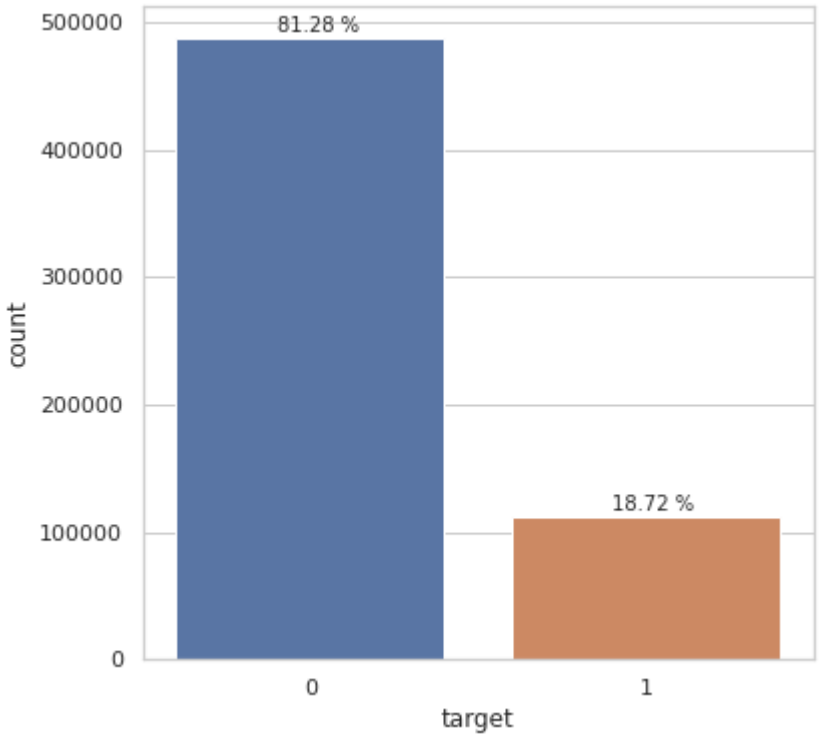
```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, cal
l drive.mount("/content/gdrive", force_remount=True).
```

### Reading Input Data

In [0]:
```
# Reading data into dataframe using pandas
df_train= pd.read_csv('/content/gdrive/My Drive/Kaggle/train.csv')
df_test=pd.read_csv(r'/content/gdrive/My Drive/Kaggle/test.csv')
```

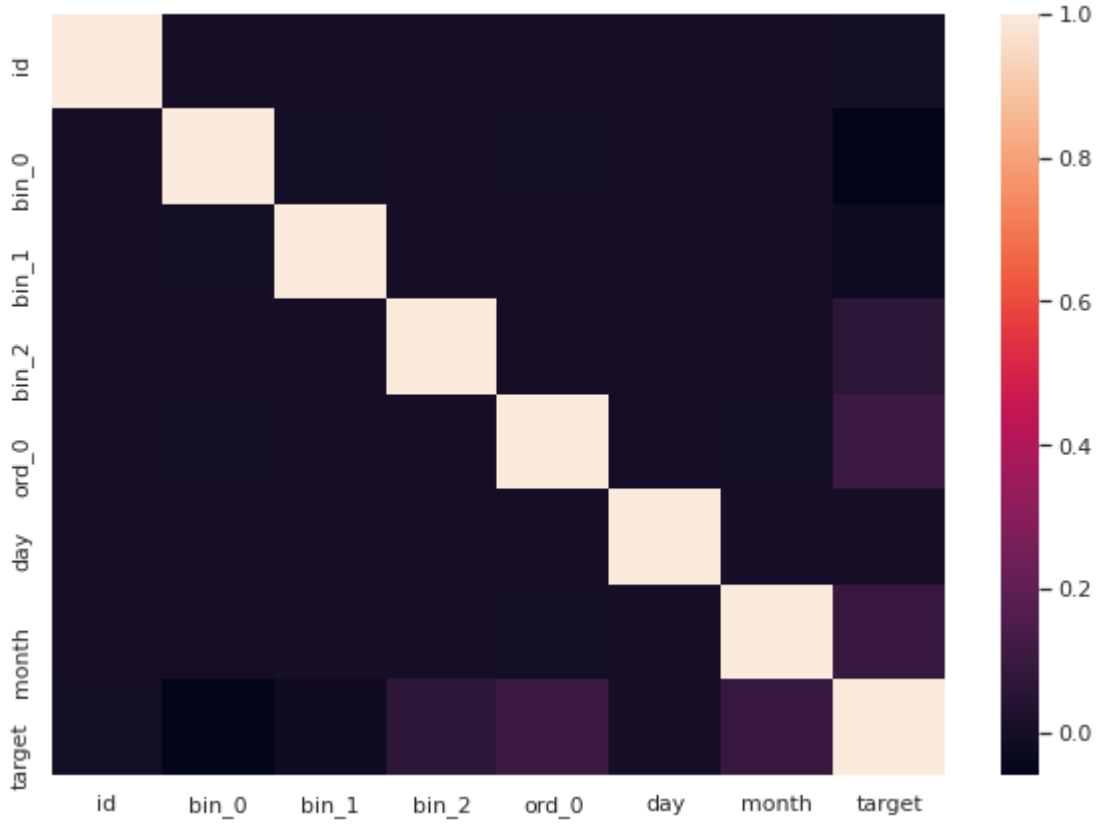
### Exploratory Data Analysis

In [4]:
```
#Distribution of Target variable
plt.figure(figsize=(6,6))
ax = sns.countplot(df_train.target)

height = sum([p.get_height() for p in ax.patches])
for p in ax.patches:
        ax.annotate(f'{100*p.get_height()/height:.2f} %', (p.get_x()+0.3, p.ge
t_height()+5000),animated=True)
```

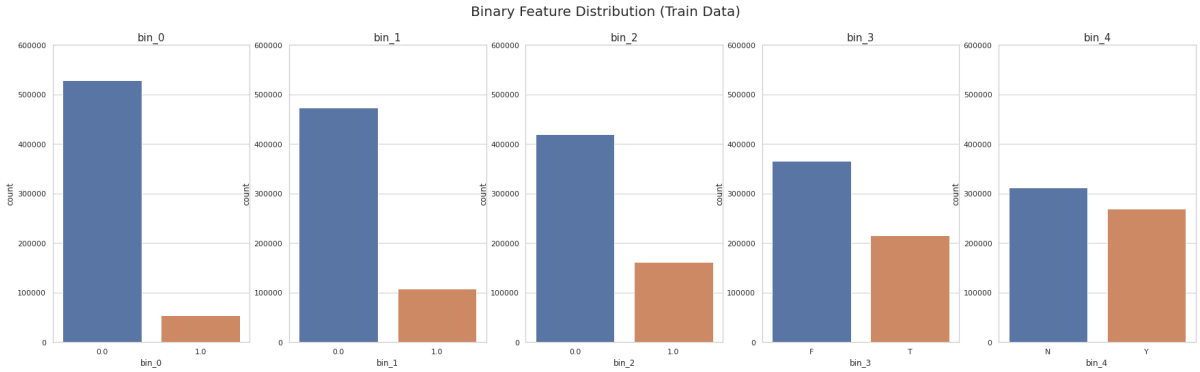


In [5]:
```
#Looking at correlations for numerical variables
plt.figure(figsize=(10,7))
num_cols = df_train.select_dtypes(exclude=['object']).columns
corr = df_train[num_cols].corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
```

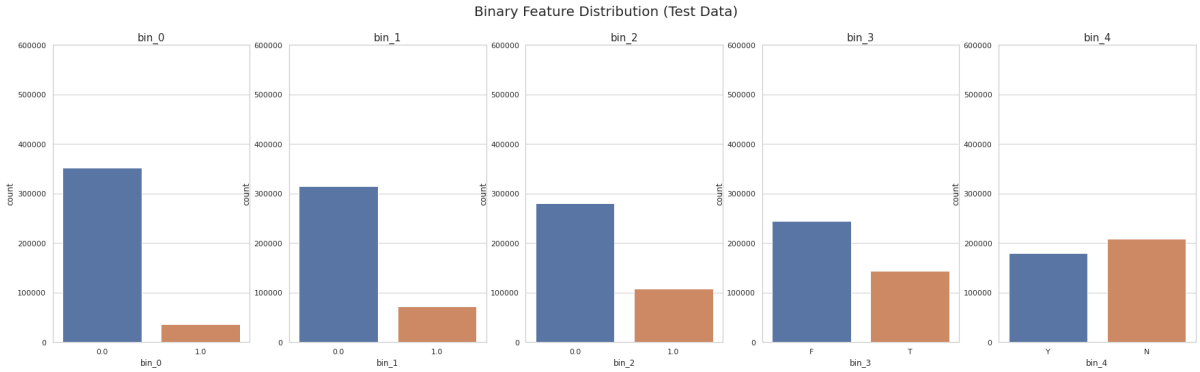
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7f08499cef28>



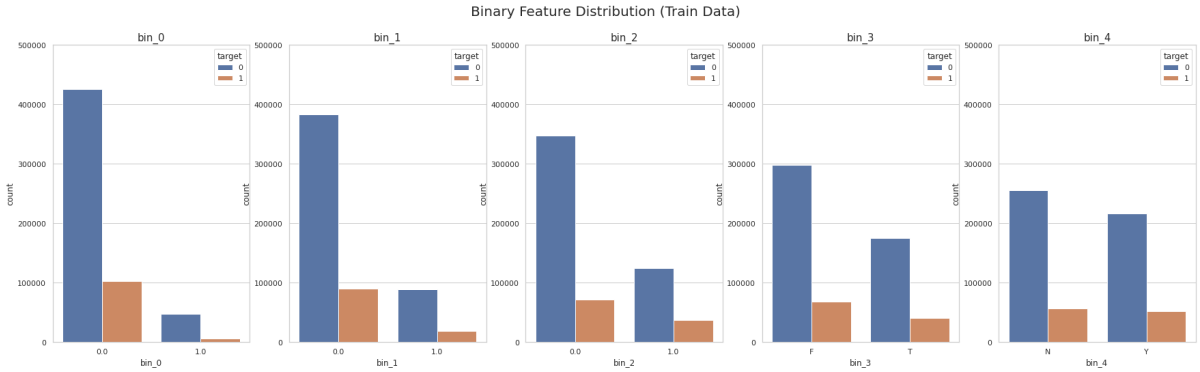

In [6]:
```
#Looking at Distribution of binary features on train data
fig, ax = plt.subplots(1,5, figsize=(30,8))
for i in range(5):
    sns.countplot(f'bin_{i}', data= df_train, ax=ax[i])
    ax[i].set_ylim([0, 600000])
    ax[i].set_title(f'bin_{i}', fontsize=15)
fig.suptitle("Binary Feature Distribution (Train Data)", fontsize=20)
plt.show()
```



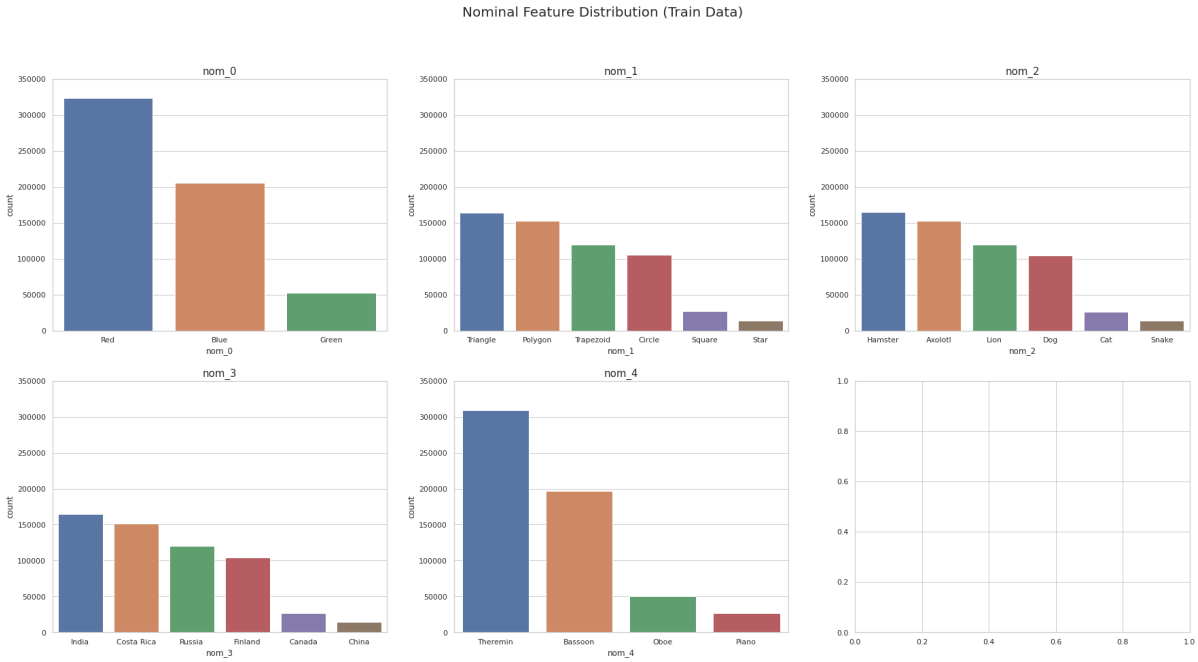
In [7]:
```
#Looking at Distribution of binary features on test data
fig, ax = plt.subplots(1,5, figsize=(30,8))
for i in range(5):
    sns.countplot(f'bin_{i}', data= df_test, ax=ax[i])
    ax[i].set_ylim([0, 600000])
    ax[i].set_title(f'bin_{i}', fontsize=15)
fig.suptitle("Binary Feature Distribution (Test Data)", fontsize=20)
plt.show()
```




In [8]:
```
#Looking at Distribution of binary features on train data
fig, ax = plt.subplots(1,5, figsize=(30, 8))
for i in range(5):
    sns.countplot(f'bin_{i}', hue='target', data= df_train, ax=ax[i])
    ax[i].set_ylim([0, 500000])
    ax[i].set_title(f'bin_{i}', fontsize=15)
fig.suptitle("Binary Feature Distribution (Train Data)", fontsize=20)
plt.show()
```

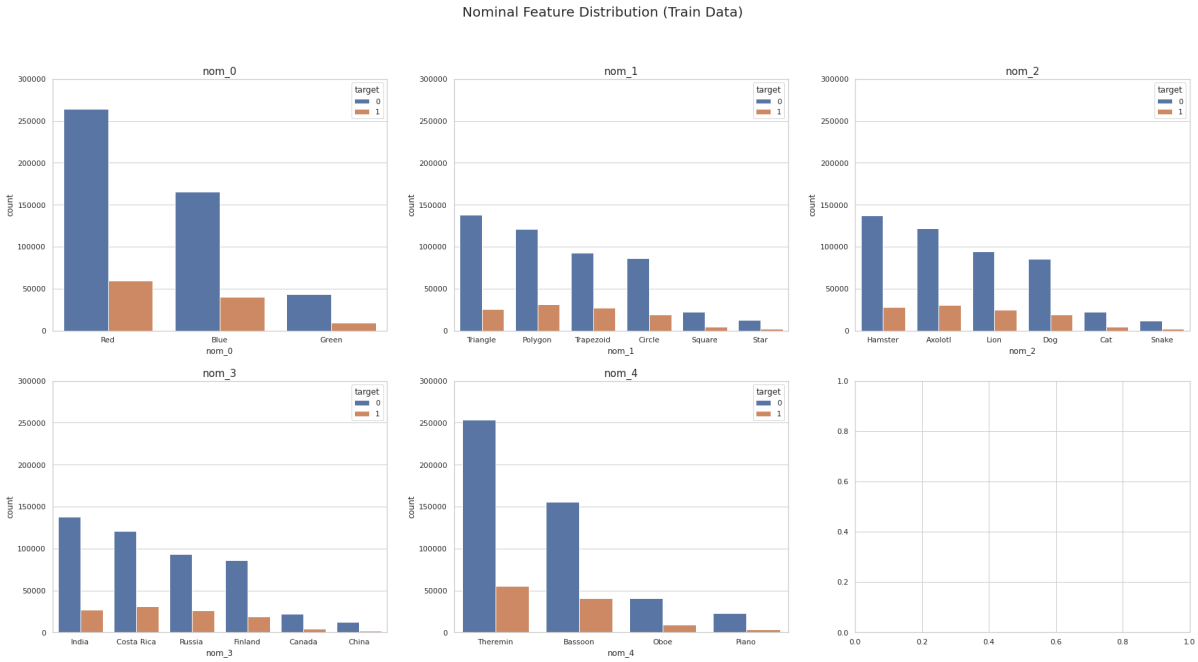


In [9]:
```
#Looking at Distribution of nominal features on train data
fig, ax = plt.subplots(2,3, figsize=(30, 15))
for i in range(5):
    sns.countplot(f'nom_{i}', data= df_train, ax=ax[i//3][i%3],
                  order=df_train[f'nom_{i}'].value_counts().index)
    ax[i//3][i%3].set_ylim([0, 350000])
    ax[i//3][i%3].set_title(f'nom_{i}', fontsize=15)
fig.suptitle("Nominal Feature Distribution (Train Data)", fontsize=20)
plt.show()
```

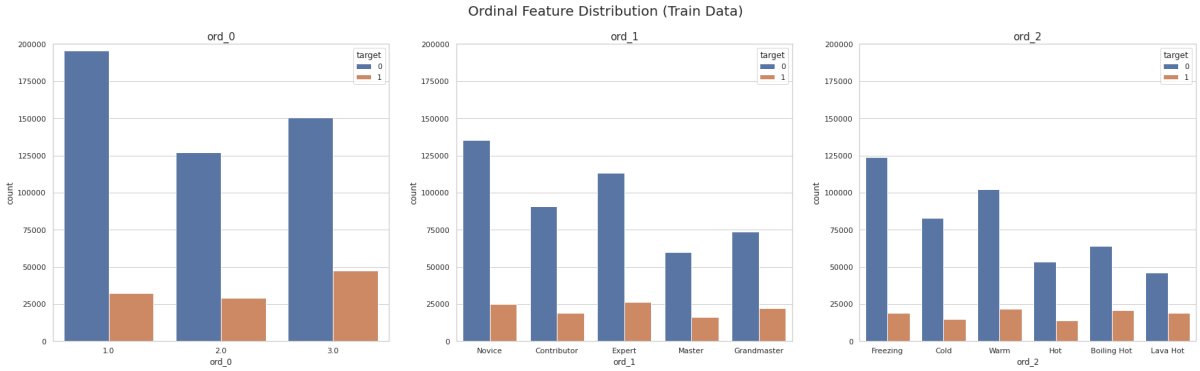


In [10]:
```
#Looking at Distribution of nominal features on test data
fig, ax = plt.subplots(2,3, figsize=(30, 15))
for i in range(5):
    sns.countplot(f'nom_{i}', data= df_test, ax=ax[i//3][i%3],
                  order=df_test[f'nom_{i}'].value_counts().index)
    ax[i//3][i%3].set_ylim([0, 350000])
    ax[i//3][i%3].set_title(f'nom_{i}', fontsize=15)
fig.suptitle("Nominal Feature Distribution (Test Data)", fontsize=20)
plt.show()
```

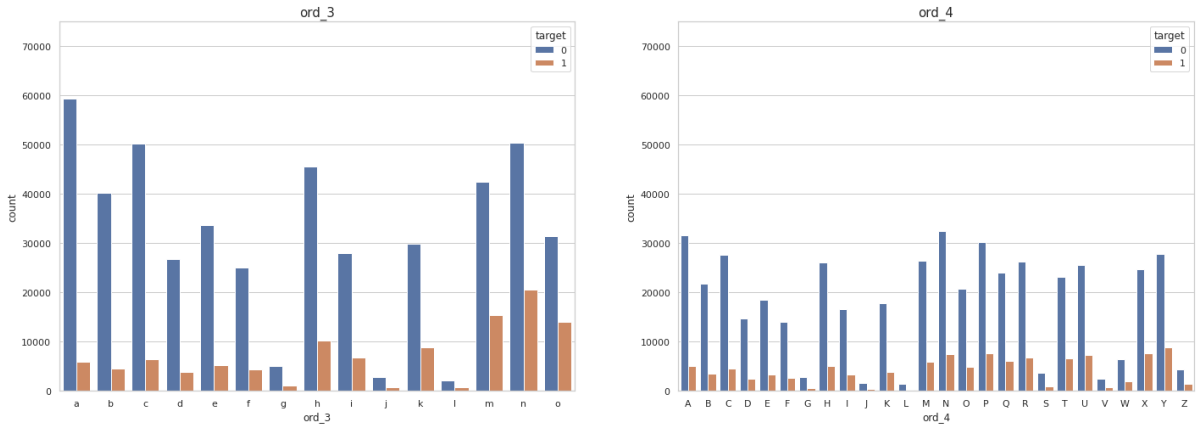
In [11]:
```
#Looking at Distribution of nominal features on train data
fig, ax = plt.subplots(2,3, figsize=(30, 15))
for i in range(5):
    sns.countplot(f'nom_{i}', hue='target', data= df_train, ax=ax[i//3][i%3],
                  order=df_train[f'nom_{i}'].value_counts().index)
    ax[i//3][i%3].set_ylim([0, 300000])
    ax[i//3][i%3].set_title(f'nom_{i}', fontsize=15)
fig.suptitle("Nominal Feature Distribution (Train Data)", fontsize=20)
plt.show()
```



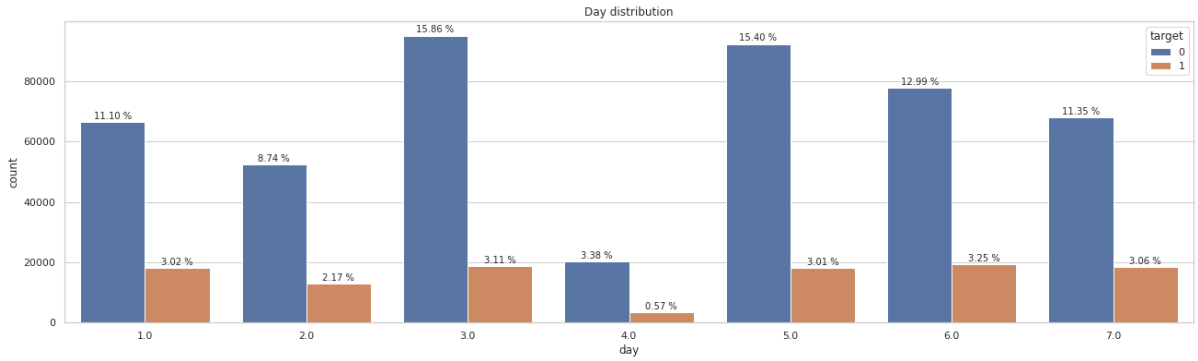
In [12]:
```
#Looking at Distribution of ordinal features on train data
fig, ax = plt.subplots(1,3, figsize=(30, 8))

ord_order = [
    [1.0, 2.0, 3.0],
    ['Novice', 'Contributor', 'Expert', 'Master', 'Grandmaster'],
    ['Freezing', 'Cold', 'Warm', 'Hot', 'Boiling Hot', 'Lava Hot']
]

for i in range(3):
    sns.countplot(f'ord_{i}', hue='target', data= df_train, ax=ax[i],
                  order = ord_order[i]
                 )
    ax[i].set_ylim([0, 200000])
    ax[i].set_title(f'ord_{i}', fontsize=15)
fig.suptitle("Ordinal Feature Distribution (Train Data)", fontsize=20)
plt.show()
```



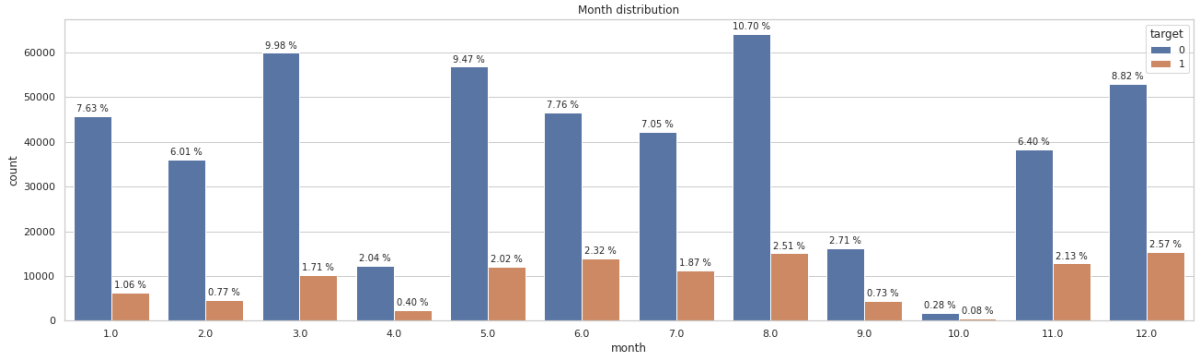

In [13]:
```
#Looking at Distribution of ordinal features on train data
fig, ax = plt.subplots(1,2, figsize=(24, 8))

for i in range(3, 5):
    sns.countplot(f'ord_{i}', hue='target', data= df_train, ax=ax[i-3],
                  order = sorted(df_train[f'ord_{i}'].dropna().unique())
                 )
    ax[i-3].set_ylim([0, 75000])
    ax[i-3].set_title(f'ord_{i}', fontsize=15)
plt.show()
```



In [14]:
```
#distribution of day varibale
plt.figure(figsize=(22,6))
plt.title('Day distribution')
ax = sns.countplot(df_train.day, hue=df_train.target)
for p in ax.patches:
    ax.text(p.get_x()+p.get_width()/2., p.get_height()+1000, f'{100*p.get_heig
ht()/height:.2f} %',ha='center')
plt.show()
```




In [15]:
```
#distribution of month varibale
plt.figure(figsize=(22,6))
plt.title('Month distribution')
ax = sns.countplot(df_train.month, hue=df_train.target)
for p in ax.patches:
    ax.text(p.get_x()+p.get_width()/2., p.get_height()+1000, f'{100*p.get_heig
ht()/height:.2f} %', ha='center')
plt.show()
```




In [16]:
```
df_train.columns
```

Out[16]:
```
Index(['id', 'bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_
9',
       'ord_0', 'ord_1', 'ord_2', 'ord_3', 'ord_4', 'ord_5', 'day', 'month',
       'target'],
      dtype='object')
```

In [17]:
```
df_test.columns
```

Out[17]:
```
Index(['id', 'bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_
9',
       'ord_0', 'ord_1', 'ord_2', 'ord_3', 'ord_4', 'ord_5', 'day', 'month'],
      dtype='object')
```

In [18]:
```
df_train.head()
```

Out[18]:

| | id | bin_0 | bin_1 | bin_2 | bin_3 | bin_4 | nom_0 | nom_1 | nom_2 | nom_3 | nom_4 | nom |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0.0 | 0.0 | 0.0 | F | N | Red | Trapezoid | Hamster | Russia | Bassoon | de4c57e |
| **1** | 1 | 1.0 | 1.0 | 0.0 | F | Y | Red | Star | Axolotl | NaN | Theremin | 2bb3c3e |
| **2** | 2 | 0.0 | 1.0 | 0.0 | F | N | Red | NaN | Hamster | Canada | Bassoon | b574c98 |
| **3** | 3 | NaN | 0.0 | 0.0 | F | N | Red | Circle | Hamster | Finland | Theremin | 673bdf |
| **4** | 4 | 0.0 | NaN | 0.0 | T | N | Red | Triangle | Hamster | Costa Rica | NaN | 777d1ac |

In [19]:
```
df_test.head()
```

Out[19]:

| | id | bin_0 | bin_1 | bin_2 | bin_3 | bin_4 | nom_0 | nom_1 | nom_2 | nom_3 | nom_4 | nc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 600000 | 0.0 | 0.0 | 0.0 | F | Y | Blue | Polygon | Axolotl | Finland | Piano | 52f6d |
| **1** | 600001 | 0.0 | 0.0 | 0.0 | F | Y | Red | Circle | Lion | Russia | Bassoon | 691eb |
| **2** | 600002 | 0.0 | 0.0 | 0.0 | F | Y | Blue | Circle | Axolotl | Russia | Theremin | 81f79 |
| **3** | 600003 | 1.0 | 0.0 | 0.0 | F | N | Red | Polygon | Axolotl | Costa Rica | Bassoon | c9134 |
| **4** | 600004 | 0.0 | 0.0 | 1.0 | F | Y | Red | Circle | NaN | Finland | Theremin | f0f1( |

In [20]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600000 entries, 0 to 599999
Data columns (total 25 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   id      600000 non-null  int64  
 1   bin_0   582106 non-null  float64
 2   bin_1   581997 non-null  float64
 3   bin_2   582070 non-null  float64
 4   bin_3   581986 non-null  object 
 5   bin_4   581953 non-null  object 
 6   nom_0   581748 non-null  object 
 7   nom_1   581844 non-null  object 
 8   nom_2   581965 non-null  object 
 9   nom_3   581879 non-null  object 
 10  nom_4   581965 non-null  object 
 11  nom_5   582222 non-null  object 
 12  nom_6   581869 non-null  object 
 13  nom_7   581997 non-null  object 
 14  nom_8   582245 non-null  object 
 15  nom_9   581927 non-null  object 
 16  ord_0   581712 non-null  float64
 17  ord_1   581959 non-null  object 
 18  ord_2   581925 non-null  object 
 19  ord_3   582084 non-null  object 
 20  ord_4   582070 non-null  object 
 21  ord_5   582287 non-null  object 
 22  day     582048 non-null  float64
 23  month   582012 non-null  float64
 24  target  600000 non-null  int64  
dtypes: float64(6), int64(2), object(17)
memory usage: 114.4+ MB
```

In [21]:
```
df_train.nunique()
```

Out[21]:
```
id        600000
bin_0          2
bin_1          2
bin_2          2
bin_3          2
bin_4          2
nom_0          3
nom_1          6
nom_2          6
nom_3          6
nom_4          4
nom_5       1220
nom_6       1519
nom_7        222
nom_8        222
nom_9       2218
ord_0          3
ord_1          5
ord_2          6
ord_3         15
ord_4         26
ord_5        190
day            7
month         12
target         2
dtype: int64
```

In [0]:
```
#Convert object to string
df_train[['bin_3','bin_4','nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9'
]]= df_train[['bin_3','bin_4','nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9'
]].astype('str')

df_test[['bin_3','bin_4','nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9'
]]= df_test[['bin_3','bin_4','nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9'
]].astype('str')
```

In [0]:
```
#Mask the Null values to retain them during encoding
mask_train= df_train.isin(['nan'])
mask_test= df_test.isin(['nan'])
```

### Encoding Nominal Variables

In [0]:
```
#Encode all the nominal features and the two string binary feature


class MultiColumnLabelEncoder:
    def __init__(self,columns = None):
        self.columns = columns # array of column names to encode

    def fit(self,X,y=None):
        return self # not relevant here

    def transform(self,X):
        '''
        Transforms columns of X specified in self.columns using
        LabelEncoder(). If no columns specified, transforms all
        columns in X.
        '''
        output = X.copy()
        if self.columns is not None:
            for col in self.columns:
                output[col] = LabelEncoder().fit_transform(output[col])
        else:
            for colname,col in output.iteritems():
                output[colname] = LabelEncoder().fit_transform(col)
        return output

    def fit_transform(self,X,y=None):
        return self.fit(X,y).transform(X)
```

In [0]:
```
df_train_enc= MultiColumnLabelEncoder(columns = [ 'bin_3','bin_4','nom_0', 'no
m_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9'
]).fit_transform(df_train)
df_train_enc=pd.DataFrame(df_train_enc.where(~mask_train, other=np.nan))

df_test_enc= MultiColumnLabelEncoder(columns = [ 'bin_3','bin_4','nom_0', 'nom
_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9'
]).fit_transform(df_test)
df_test_enc=pd.DataFrame(df_test_enc.where(~mask_test, other=np.nan))
```

**Encoding Ordinal Variables**

In [0]:
```
#map low cardinality ordinal features

map_ord1 = {'Novice':1, 'Contributor':2, 'Expert':3, 'Master':4, 'Grandmaster'
:5}

df_train_enc.ord_1 = df_train_enc.ord_1.replace(map_ord1)
df_test_enc.ord_1 = df_test_enc.ord_1.replace(map_ord1)

map_ord2 = {'Freezing':1, 'Cold':10, 'Warm':25, 'Hot':50, 'Boiling Hot':100,
'Lava Hot':800}

df_train_enc.ord_2 = df_train_enc.ord_2.replace(map_ord2)
df_test_enc.ord_2 = df_test_enc.ord_2.replace(map_ord2)
```

In [0]:
```
#Encode high cardinality features
map_ord3 = {'a':1, 'b':2, 'c':3, 'd':4, 'e':5, 'f':6, 'g':7, 'h':8, 'i':9, 'j'
:10, 'k':11, 'l':12,'m':13,'n':14, 'o':15}
df_train_enc.ord_3 = df_train_enc.ord_3.replace(map_ord3)
df_test_enc.ord_3 = df_test_enc.ord_3.replace(map_ord3)

map_ord4 = {'A':1, 'B':2, 'C':3, 'D':4, 'E':5, 'F':6, 'G':7, 'H':8, 'I':9, 'J'
:10, 'K':11, 'L':12,'M':13,'N':14, 'O':15,
           'P':16, 'Q':17, 'R':18, 'S':19, 'T':20, 'U':21, 'V':22, 'W':23, 'X'
:24, 'Y':25, 'Z':26}
df_train_enc.ord_4 = df_train_enc.ord_4.replace(map_ord4)
df_test_enc.ord_4 = df_test_enc.ord_4.replace(map_ord4)
```

In [0]:
```
df_train_enc['ord_5_enc']=df_train_enc['ord_5']
df_test_enc['ord_5_enc']=df_test_enc['ord_5']

ce_ord = ce.OrdinalEncoder(cols = ['ord_5'])

df_train_encall=ce_ord.fit_transform(df_train_enc, df_train_enc['ord_5_enc'])
df_test_encall=ce_ord.fit_transform(df_test_enc, df_test_enc['ord_5_enc'])
```

In [29]:
```
df_train_enc.columns
```

Out[29]:
```
Index(['id', 'bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_
9',
       'ord_0', 'ord_1', 'ord_2', 'ord_3', 'ord_4', 'ord_5', 'day', 'month',
       'target', 'ord_5_enc'],
      dtype='object')
```

In [0]:
```
df_train_encoded= df_train_encall.drop(['target','ord_5_enc'], axis=1)
df_test_encoded= df_test_encall.drop('ord_5_enc', axis=1)
```

In [31]:
```
df_train_encoded.columns
```

Out[31]:
```
Index(['id', 'bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_
9',
       'ord_0', 'ord_1', 'ord_2', 'ord_3', 'ord_4', 'ord_5', 'day', 'month'],
      dtype='object')
```

In [32]:
```
df_train_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 600000 entries, 0 to 599999
Data columns (total 24 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   id      600000 non-null  int64  
 1   bin_0   582106 non-null  float64
 2   bin_1   581997 non-null  float64
 3   bin_2   582070 non-null  float64
 4   bin_3   581986 non-null  float64
 5   bin_4   581953 non-null  float64
 6   nom_0   581748 non-null  float64
 7   nom_1   581844 non-null  float64
 8   nom_2   581965 non-null  float64
 9   nom_3   581879 non-null  float64
 10  nom_4   581965 non-null  float64
 11  nom_5   582222 non-null  float64
 12  nom_6   581869 non-null  float64
 13  nom_7   581997 non-null  float64
 14  nom_8   582245 non-null  float64
 15  nom_9   581927 non-null  float64
 16  ord_0   581712 non-null  float64
 17  ord_1   581959 non-null  float64
 18  ord_2   581925 non-null  float64
 19  ord_3   582084 non-null  float64
 20  ord_4   582070 non-null  float64
 21  ord_5   600000 non-null  int64  
 22  day     582048 non-null  float64
 23  month   582012 non-null  float64
dtypes: float64(22), int64(2)
memory usage: 109.9 MB
```

In [33]:
```
df_test_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400000 entries, 0 to 399999
Data columns (total 24 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   id      400000 non-null  int64  
 1   bin_0   388099 non-null  float64
 2   bin_1   387962 non-null  float64
 3   bin_2   388028 non-null  float64
 4   bin_3   388049 non-null  float64
 5   bin_4   388049 non-null  float64
 6   nom_0   387938 non-null  float64
 7   nom_1   388053 non-null  float64
 8   nom_2   387821 non-null  float64
 9   nom_3   387824 non-null  float64
 10  nom_4   388007 non-null  float64
 11  nom_5   388088 non-null  float64
 12  nom_6   387988 non-null  float64
 13  nom_7   387997 non-null  float64
 14  nom_8   388044 non-null  float64
 15  nom_9   387940 non-null  float64
 16  ord_0   388107 non-null  float64
 17  ord_1   387833 non-null  float64
 18  ord_2   387895 non-null  float64
 19  ord_3   387947 non-null  float64
 20  ord_4   388067 non-null  float64
 21  ord_5   400000 non-null  int64  
 22  day     387975 non-null  float64
 23  month   388016 non-null  float64
dtypes: float64(22), int64(2)
memory usage: 73.2 MB
```

### Missing Imputing Values

In [0]:
```
#for median imputation replace 'mean' with 'median'
imp_mean = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imp_mean.fit(df_train_encoded)
df_train_imputed=pd.DataFrame(imp_mean.transform(df_train_encoded))
```

In [0]:
```
imp_mean2 = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
imp_mean2.fit(df_test_encoded)
df_test_imputed=pd.DataFrame(imp_mean.transform(df_test_encoded))
```

In [0]:
```
df_train_imputed.columns=df_train_encoded.columns
df_test_imputed.columns=df_test_encoded.columns
```

In [37]:
```
print(df_train_imputed.dtypes)
```

```
id       float64
bin_0    float64
bin_1    float64
bin_2    float64
bin_3    float64
bin_4    float64
nom_0    float64
nom_1    float64
nom_2    float64
nom_3    float64
nom_4    float64
nom_5    float64
nom_6    float64
nom_7    float64
nom_8    float64
nom_9    float64
ord_0    float64
ord_1    float64
ord_2    float64
ord_3    float64
ord_4    float64
ord_5    float64
day      float64
month    float64
dtype: object
```

In [38]:
```
print(df_test_imputed.dtypes)
```

```
id       float64
bin_0    float64
bin_1    float64
bin_2    float64
bin_3    float64
bin_4    float64
nom_0    float64
nom_1    float64
nom_2    float64
nom_3    float64
nom_4    float64
nom_5    float64
nom_6    float64
nom_7    float64
nom_8    float64
nom_9    float64
ord_0    float64
ord_1    float64
ord_2    float64
ord_3    float64
ord_4    float64
ord_5    float64
day      float64
month    float64
dtype: object
```

### Spliting Data to Train and Test

In [0]:
```
X_train, X_val, y_train, y_val = train_test_split(df_train_imputed,df_train_en
call.target, test_size=0.25, random_state=420)
```

In [40]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 450000 entries, 334543 to 193608
Data columns (total 24 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   id      450000 non-null  float64
 1   bin_0   450000 non-null  float64
 2   bin_1   450000 non-null  float64
 3   bin_2   450000 non-null  float64
 4   bin_3   450000 non-null  float64
 5   bin_4   450000 non-null  float64
 6   nom_0   450000 non-null  float64
 7   nom_1   450000 non-null  float64
 8   nom_2   450000 non-null  float64
 9   nom_3   450000 non-null  float64
 10  nom_4   450000 non-null  float64
 11  nom_5   450000 non-null  float64
 12  nom_6   450000 non-null  float64
 13  nom_7   450000 non-null  float64
 14  nom_8   450000 non-null  float64
 15  nom_9   450000 non-null  float64
 16  ord_0   450000 non-null  float64
 17  ord_1   450000 non-null  float64
 18  ord_2   450000 non-null  float64
 19  ord_3   450000 non-null  float64
 20  ord_4   450000 non-null  float64
 21  ord_5   450000 non-null  float64
 22  day     450000 non-null  float64
 23  month   450000 non-null  float64
dtypes: float64(24)
memory usage: 85.8 MB
```

In [41]:
```
print(X_train.shape)
print(X_val.shape)
print(X_val.columns)
```

```
(450000, 24)
(150000, 24)
Index(['id', 'bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_
9',
       'ord_0', 'ord_1', 'ord_2', 'ord_3', 'ord_4', 'ord_5', 'day', 'month'],
      dtype='object')
```

In [42]:
```
print(df_test.shape)
print(df_test.columns)
```

```
(400000, 24)
Index(['id', 'bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'nom_0', 'nom_1',
       'nom_2', 'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_
9',
       'ord_0', 'ord_1', 'ord_2', 'ord_3', 'ord_4', 'ord_5', 'day', 'month'],
      dtype='object')
```

In [43]:
```
train= X_train.drop('id',axis=1)
val= X_val.drop('id',axis=1)
test= df_test_imputed.drop('id',axis=1)
print(train.shape)
print(val.shape)
print(test.shape)
print(train.columns)
```

```
(450000, 23)
(150000, 23)
(400000, 23)
Index(['bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'nom_0', 'nom_1', 'nom_
2',
       'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9', 'ord_
0',
       'ord_1', 'ord_2', 'ord_3', 'ord_4', 'ord_5', 'day', 'month'],
      dtype='object')
```

In [44]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400000 entries, 0 to 399999
Data columns (total 23 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   bin_0   400000 non-null  float64
 1   bin_1   400000 non-null  float64
 2   bin_2   400000 non-null  float64
 3   bin_3   400000 non-null  float64
 4   bin_4   400000 non-null  float64
 5   nom_0   400000 non-null  float64
 6   nom_1   400000 non-null  float64
 7   nom_2   400000 non-null  float64
 8   nom_3   400000 non-null  float64
 9   nom_4   400000 non-null  float64
 10  nom_5   400000 non-null  float64
 11  nom_6   400000 non-null  float64
 12  nom_7   400000 non-null  float64
 13  nom_8   400000 non-null  float64
 14  nom_9   400000 non-null  float64
 15  ord_0   400000 non-null  float64
 16  ord_1   400000 non-null  float64
 17  ord_2   400000 non-null  float64
 18  ord_3   400000 non-null  float64
 19  ord_4   400000 non-null  float64
 20  ord_5   400000 non-null  float64
 21  day     400000 non-null  float64
 22  month   400000 non-null  float64
dtypes: float64(23)
memory usage: 70.2 MB
```

### Standarizing the features

In [0]:
```
from sklearn.preprocessing import StandardScaler
SC= StandardScaler()
train_sc = SC.fit_transform(train)
val_sc=SC.fit_transform(val)
test_sc=SC.fit_transform(test)
```

### Dealing with Data imbalance

In [0]:
```
weight = float(len(y_train[y_train == 0]))/float(len(y_train[y_train == 1]))
w1 = np.array([1]*y_train.shape[0])
w1[y_train==1]=weight
```

### Model Building

In [0]:
```
model = XGBClassifier(objective ='binary:logistic',
                      colsample_bytree = 0,
                      learning_rate = 0.2,
                      max_depth = 15,
                      n_estimators = 400,
                      scale_pos_weight = 2,
                      random_state = 2020,
                      subsample = 0.8)
```

In [48]:
```
model.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=True, sample_we
ight=w1)
```

```
[0]     validation_0-error:0.807167
[1]     validation_0-error:0.81216
[2]     validation_0-error:0.81298
[3]     validation_0-error:0.81318
[4]     validation_0-error:0.813513
[5]     validation_0-error:0.813587
[6]     validation_0-error:0.8136
[7]     validation_0-error:0.81346
[8]     validation_0-error:0.812127
[9]     validation_0-error:0.811193
[10]    validation_0-error:0.811767
[11]    validation_0-error:0.809593
[12]    validation_0-error:0.809347
[13]    validation_0-error:0.806933
[14]    validation_0-error:0.805533
[15]    validation_0-error:0.804833
[16]    validation_0-error:0.805287
[17]    validation_0-error:0.79248
[18]    validation_0-error:0.784353
[19]    validation_0-error:0.784747
[20]    validation_0-error:0.784953
[21]    validation_0-error:0.78524
[22]    validation_0-error:0.785467
[23]    validation_0-error:0.785567
[24]    validation_0-error:0.783067
[25]    validation_0-error:0.776627
[26]    validation_0-error:0.77504
[27]    validation_0-error:0.773353
[28]    validation_0-error:0.765933
[29]    validation_0-error:0.761753
[30]    validation_0-error:0.75572
[31]    validation_0-error:0.75262
[32]    validation_0-error:0.75078
[33]    validation_0-error:0.74744
[34]    validation_0-error:0.747113
[35]    validation_0-error:0.743907
[36]    validation_0-error:0.733993
[37]    validation_0-error:0.732113
[38]    validation_0-error:0.7291
[39]    validation_0-error:0.719067
[40]    validation_0-error:0.71666
[41]    validation_0-error:0.713393
[42]    validation_0-error:0.7114
[43]    validation_0-error:0.710673
[44]    validation_0-error:0.708733
[45]    validation_0-error:0.706407
[46]    validation_0-error:0.705787
[47]    validation_0-error:0.705733
[48]    validation_0-error:0.70448
[49]    validation_0-error:0.697993
[50]    validation_0-error:0.69272
[51]    validation_0-error:0.69056
[52]    validation_0-error:0.6906
[53]    validation_0-error:0.683487
[54]    validation_0-error:0.660447
[55]    validation_0-error:0.66048
[56]    validation_0-error:0.659327
```

```
[57]    validation_0-error:0.65496
[58]    validation_0-error:0.647207
[59]    validation_0-error:0.645047
[60]    validation_0-error:0.644087
[61]    validation_0-error:0.644013
[62]    validation_0-error:0.63796
[63]    validation_0-error:0.637927
[64]    validation_0-error:0.637787
[65]    validation_0-error:0.630933
[66]    validation_0-error:0.626147
[67]    validation_0-error:0.626227
[68]    validation_0-error:0.621633
[69]    validation_0-error:0.6217
[70]    validation_0-error:0.619833
[71]    validation_0-error:0.617913
[72]    validation_0-error:0.61604
[73]    validation_0-error:0.613307
[74]    validation_0-error:0.612267
[75]    validation_0-error:0.611207
[76]    validation_0-error:0.610793
[77]    validation_0-error:0.606647
[78]    validation_0-error:0.6018
[79]    validation_0-error:0.601607
[80]    validation_0-error:0.601193
[81]    validation_0-error:0.593267
[82]    validation_0-error:0.593407
[83]    validation_0-error:0.593673
[84]    validation_0-error:0.593627
[85]    validation_0-error:0.589027
[86]    validation_0-error:0.588133
[87]    validation_0-error:0.58722
[88]    validation_0-error:0.585393
[89]    validation_0-error:0.584693
[90]    validation_0-error:0.584487
[91]    validation_0-error:0.58462
[92]    validation_0-error:0.584113
[93]    validation_0-error:0.5826
[94]    validation_0-error:0.58084
[95]    validation_0-error:0.580127
[96]    validation_0-error:0.577507
[97]    validation_0-error:0.571093
[98]    validation_0-error:0.570153
[99]    validation_0-error:0.569107
[100]   validation_0-error:0.567733
[101]   validation_0-error:0.567047
[102]   validation_0-error:0.56572
[103]   validation_0-error:0.562433
[104]   validation_0-error:0.561967
[105]   validation_0-error:0.5605
[106]   validation_0-error:0.558307
[107]   validation_0-error:0.55838
[108]   validation_0-error:0.557833
[109]   validation_0-error:0.55578
[110]   validation_0-error:0.5536
[111]   validation_0-error:0.549067
[112]   validation_0-error:0.54832
[113]   validation_0-error:0.547373
```

```
[114]   validation_0-error:0.54704
[115]   validation_0-error:0.53714
[116]   validation_0-error:0.529233
[117]   validation_0-error:0.52876
[118]   validation_0-error:0.5282
[119]   validation_0-error:0.527207
[120]   validation_0-error:0.52064
[121]   validation_0-error:0.517913
[122]   validation_0-error:0.51324
[123]   validation_0-error:0.512913
[124]   validation_0-error:0.51146
[125]   validation_0-error:0.51044
[126]   validation_0-error:0.510187
[127]   validation_0-error:0.5092
[128]   validation_0-error:0.508573
[129]   validation_0-error:0.50546
[130]   validation_0-error:0.50486
[131]   validation_0-error:0.505013
[132]   validation_0-error:0.504787
[133]   validation_0-error:0.504487
[134]   validation_0-error:0.50438
[135]   validation_0-error:0.503847
[136]   validation_0-error:0.50358
[137]   validation_0-error:0.503813
[138]   validation_0-error:0.5034
[139]   validation_0-error:0.501387
[140]   validation_0-error:0.500553
[141]   validation_0-error:0.497867
[142]   validation_0-error:0.497753
[143]   validation_0-error:0.497387
[144]   validation_0-error:0.4971
[145]   validation_0-error:0.49666
[146]   validation_0-error:0.496453
[147]   validation_0-error:0.495987
[148]   validation_0-error:0.495787
[149]   validation_0-error:0.492107
[150]   validation_0-error:0.49176
[151]   validation_0-error:0.489667
[152]   validation_0-error:0.489647
[153]   validation_0-error:0.489173
[154]   validation_0-error:0.48826
[155]   validation_0-error:0.488067
[156]   validation_0-error:0.486807
[157]   validation_0-error:0.485707
[158]   validation_0-error:0.484893
[159]   validation_0-error:0.48498
[160]   validation_0-error:0.484553
[161]   validation_0-error:0.484593
[162]   validation_0-error:0.48442
[163]   validation_0-error:0.48428
[164]   validation_0-error:0.484153
[165]   validation_0-error:0.48386
[166]   validation_0-error:0.482833
[167]   validation_0-error:0.48244
[168]   validation_0-error:0.481987
[169]   validation_0-error:0.481267
[170]   validation_0-error:0.48144
```

```
[171]   validation_0-error:0.479853
[172]   validation_0-error:0.479973
[173]   validation_0-error:0.479873
[174]   validation_0-error:0.479673
[175]   validation_0-error:0.478673
[176]   validation_0-error:0.478253
[177]   validation_0-error:0.47756
[178]   validation_0-error:0.477567
[179]   validation_0-error:0.477253
[180]   validation_0-error:0.47694
[181]   validation_0-error:0.476093
[182]   validation_0-error:0.47566
[183]   validation_0-error:0.474727
[184]   validation_0-error:0.474
[185]   validation_0-error:0.4741
[186]   validation_0-error:0.47426
[187]   validation_0-error:0.473667
[188]   validation_0-error:0.47316
[189]   validation_0-error:0.472907
[190]   validation_0-error:0.472133
[191]   validation_0-error:0.471247
[192]   validation_0-error:0.471307
[193]   validation_0-error:0.471347
[194]   validation_0-error:0.47124
[195]   validation_0-error:0.46904
[196]   validation_0-error:0.468767
[197]   validation_0-error:0.46862
[198]   validation_0-error:0.468787
[199]   validation_0-error:0.468187
[200]   validation_0-error:0.468007
[201]   validation_0-error:0.468167
[202]   validation_0-error:0.467947
[203]   validation_0-error:0.466833
[204]   validation_0-error:0.46616
[205]   validation_0-error:0.46594
[206]   validation_0-error:0.465933
[207]   validation_0-error:0.46566
[208]   validation_0-error:0.465207
[209]   validation_0-error:0.464953
[210]   validation_0-error:0.46492
[211]   validation_0-error:0.464787
[212]   validation_0-error:0.464787
[213]   validation_0-error:0.464007
[214]   validation_0-error:0.464113
[215]   validation_0-error:0.463353
[216]   validation_0-error:0.463413
[217]   validation_0-error:0.462273
[218]   validation_0-error:0.46206
[219]   validation_0-error:0.46216
[220]   validation_0-error:0.46222
[221]   validation_0-error:0.462013
[222]   validation_0-error:0.4608
[223]   validation_0-error:0.460613
[224]   validation_0-error:0.4604
[225]   validation_0-error:0.4602
[226]   validation_0-error:0.460187
[227]   validation_0-error:0.460313
```

```
[228]   validation_0-error:0.459947
[229]   validation_0-error:0.457553
[230]   validation_0-error:0.457253
[231]   validation_0-error:0.457353
[232]   validation_0-error:0.456653
[233]   validation_0-error:0.456667
[234]   validation_0-error:0.456633
[235]   validation_0-error:0.45672
[236]   validation_0-error:0.456873
[237]   validation_0-error:0.45708
[238]   validation_0-error:0.456647
[239]   validation_0-error:0.45562
[240]   validation_0-error:0.45492
[241]   validation_0-error:0.454393
[242]   validation_0-error:0.454187
[243]   validation_0-error:0.453913
[244]   validation_0-error:0.453693
[245]   validation_0-error:0.453587
[246]   validation_0-error:0.453293
[247]   validation_0-error:0.453107
[248]   validation_0-error:0.452953
[249]   validation_0-error:0.453073
[250]   validation_0-error:0.4528
[251]   validation_0-error:0.452767
[252]   validation_0-error:0.452767
[253]   validation_0-error:0.452653
[254]   validation_0-error:0.451853
[255]   validation_0-error:0.45196
[256]   validation_0-error:0.45184
[257]   validation_0-error:0.451767
[258]   validation_0-error:0.452007
[259]   validation_0-error:0.451647
[260]   validation_0-error:0.45164
[261]   validation_0-error:0.450927
[262]   validation_0-error:0.45078
[263]   validation_0-error:0.450207
[264]   validation_0-error:0.450487
[265]   validation_0-error:0.450327
[266]   validation_0-error:0.450227
[267]   validation_0-error:0.450327
[268]   validation_0-error:0.450333
[269]   validation_0-error:0.4503
[270]   validation_0-error:0.450007
[271]   validation_0-error:0.44986
[272]   validation_0-error:0.449953
[273]   validation_0-error:0.45
[274]   validation_0-error:0.44932
[275]   validation_0-error:0.44854
[276]   validation_0-error:0.448387
[277]   validation_0-error:0.44852
[278]   validation_0-error:0.447987
[279]   validation_0-error:0.4478
[280]   validation_0-error:0.447847
[281]   validation_0-error:0.447933
[282]   validation_0-error:0.447833
[283]   validation_0-error:0.447913
[284]   validation_0-error:0.447553
```

```
[285]   validation_0-error:0.446573
[286]   validation_0-error:0.44682
[287]   validation_0-error:0.446347
[288]   validation_0-error:0.446353
[289]   validation_0-error:0.444593
[290]   validation_0-error:0.444533
[291]   validation_0-error:0.444573
[292]   validation_0-error:0.444167
[293]   validation_0-error:0.443793
[294]   validation_0-error:0.44336
[295]   validation_0-error:0.443553
[296]   validation_0-error:0.44304
[297]   validation_0-error:0.44248
[298]   validation_0-error:0.4425
[299]   validation_0-error:0.442407
[300]   validation_0-error:0.442233
[301]   validation_0-error:0.44222
[302]   validation_0-error:0.442253
[303]   validation_0-error:0.441773
[304]   validation_0-error:0.441647
[305]   validation_0-error:0.441493
[306]   validation_0-error:0.44158
[307]   validation_0-error:0.441393
[308]   validation_0-error:0.441227
[309]   validation_0-error:0.44112
[310]   validation_0-error:0.440927
[311]   validation_0-error:0.441033
[312]   validation_0-error:0.441213
[313]   validation_0-error:0.441073
[314]   validation_0-error:0.441047
[315]   validation_0-error:0.440987
[316]   validation_0-error:0.440827
[317]   validation_0-error:0.440947
[318]   validation_0-error:0.440847
[319]   validation_0-error:0.440647
[320]   validation_0-error:0.440767
[321]   validation_0-error:0.440567
[322]   validation_0-error:0.440607
[323]   validation_0-error:0.44066
[324]   validation_0-error:0.44074
[325]   validation_0-error:0.440667
[326]   validation_0-error:0.440467
[327]   validation_0-error:0.440167
[328]   validation_0-error:0.440153
[329]   validation_0-error:0.44008
[330]   validation_0-error:0.43998
[331]   validation_0-error:0.439553
[332]   validation_0-error:0.439547
[333]   validation_0-error:0.43904
[334]   validation_0-error:0.43734
[335]   validation_0-error:0.43738
[336]   validation_0-error:0.437533
[337]   validation_0-error:0.437693
[338]   validation_0-error:0.437327
[339]   validation_0-error:0.4371
[340]   validation_0-error:0.43712
[341]   validation_0-error:0.436813
```

```
[342]   validation_0-error:0.436753
[343]   validation_0-error:0.436273
[344]   validation_0-error:0.436093
[345]   validation_0-error:0.436187
[346]   validation_0-error:0.43614
[347]   validation_0-error:0.435727
[348]   validation_0-error:0.43568
[349]   validation_0-error:0.435713
[350]   validation_0-error:0.435753
[351]   validation_0-error:0.435793
[352]   validation_0-error:0.43562
[353]   validation_0-error:0.435373
[354]   validation_0-error:0.435353
[355]   validation_0-error:0.435293
[356]   validation_0-error:0.435567
[357]   validation_0-error:0.43552
[358]   validation_0-error:0.435433
[359]   validation_0-error:0.434967
[360]   validation_0-error:0.435113
[361]   validation_0-error:0.435073
[362]   validation_0-error:0.434827
[363]   validation_0-error:0.4345
[364]   validation_0-error:0.434473
[365]   validation_0-error:0.434207
[366]   validation_0-error:0.434127
[367]   validation_0-error:0.434033
[368]   validation_0-error:0.43398
[369]   validation_0-error:0.434027
[370]   validation_0-error:0.433847
[371]   validation_0-error:0.433167
[372]   validation_0-error:0.433107
[373]   validation_0-error:0.432973
[374]   validation_0-error:0.432847
[375]   validation_0-error:0.432633
[376]   validation_0-error:0.432673
[377]   validation_0-error:0.432627
[378]   validation_0-error:0.43234
[379]   validation_0-error:0.432073
[380]   validation_0-error:0.432173
[381]   validation_0-error:0.432093
[382]   validation_0-error:0.432067
[383]   validation_0-error:0.432293
[384]   validation_0-error:0.43196
[385]   validation_0-error:0.431873
[386]   validation_0-error:0.431667
[387]   validation_0-error:0.431273
[388]   validation_0-error:0.43118
[389]   validation_0-error:0.431113
[390]   validation_0-error:0.431047
[391]   validation_0-error:0.430573
[392]   validation_0-error:0.430667
[393]   validation_0-error:0.4306
[394]   validation_0-error:0.43056
[395]   validation_0-error:0.43054
[396]   validation_0-error:0.430427
[397]   validation_0-error:0.43026
```

```
[398]   validation_0-error:0.430033
[399]   validation_0-error:0.429947
```

Out[48]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0, gamma=0,
              learning_rate=0.2, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=None, n_estimators=400, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=2020,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=2, seed=None,
              silent=None, subsample=0.8, verbosity=1)
```

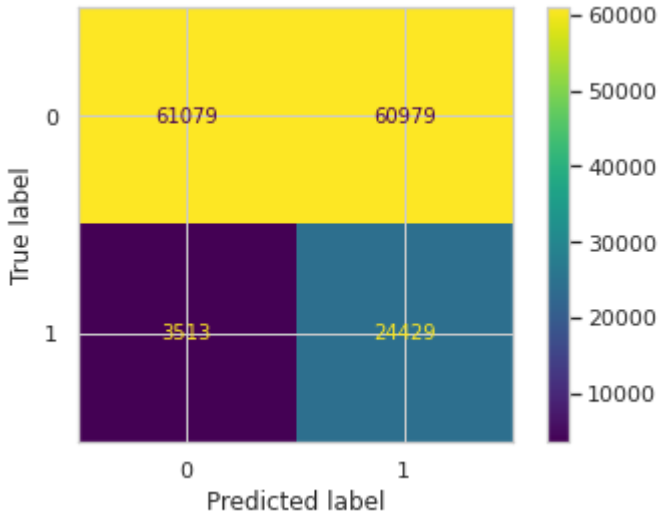
In [0]:
```
preds_val = model.predict_proba(X_val)[:,1]
```

In [50]:
```
score = metrics.roc_auc_score(y_val ,preds_val)
print("score: %f" % (score))
```

```
score: 0.780767
```

In [51]:
```
metrics.plot_confusion_matrix(model,
                              X_val, y_val,
                              normalize=None,
                              values_format='d')
```

Out[51]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f0846a10
908>
```



In [52]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400000 entries, 0 to 399999
Data columns (total 24 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   id      400000 non-null  int64  
 1   bin_0   388099 non-null  float64
 2   bin_1   387962 non-null  float64
 3   bin_2   388028 non-null  float64
 4   bin_3   400000 non-null  object 
 5   bin_4   400000 non-null  object 
 6   nom_0   400000 non-null  object 
 7   nom_1   400000 non-null  object 
 8   nom_2   400000 non-null  object 
 9   nom_3   400000 non-null  object 
 10  nom_4   400000 non-null  object 
 11  nom_5   400000 non-null  object 
 12  nom_6   400000 non-null  object 
 13  nom_7   400000 non-null  object 
 14  nom_8   400000 non-null  object 
 15  nom_9   400000 non-null  object 
 16  ord_0   388107 non-null  float64
 17  ord_1   387833 non-null  object 
 18  ord_2   387895 non-null  object 
 19  ord_3   387947 non-null  object 
 20  ord_4   388067 non-null  object 
 21  ord_5   387953 non-null  object 
 22  day     387975 non-null  float64
 23  month   388016 non-null  float64
dtypes: float64(6), int64(1), object(17)
memory usage: 73.2+ MB
```

In [0]:
```
y_test = model.predict_proba(df_test_encoded)[:,1]
```

In [54]:
```
y_test.shape
```

Out[54]:
```
(400000,)
```

In [0]:
```
samp_subm = pd.read_csv('/content/gdrive/My Drive/Kaggle/sample_submission.csv', index_col=0)
```

In [0]:
```
num = samp_subm.index
output = pd.DataFrame({'id': num,
                       'target': y_test})
```

In [57]:
```
output.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400000 entries, 0 to 399999
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   id      400000 non-null  int64  
 1   target  400000 non-null  float32
dtypes: float32(1), int64(1)
memory usage: 4.6 MB
```

In [0]:
```
output.to_csv('submission.csv', index=False)
```