CASE STUDY: CODED BY SUMEET ROUTRAY

IMPORT LIBRARIES

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
from datetime import datetime #import dattime libraries
```

READING INPUTA DATA

In [3]:
```
df1 = pd.read_csv("TrainingData_V1.csv")
df2 = pd.read_excel("TestingData_For_Candidate.xlsx", sheet_name= "Shee
t1")
```

In [4]:
```
#Creating an indicator for train-test split
df1["train_test_indicator"]="train"
df2["train_test_indicator"]="test"
```

In [5]:
```
for col in ['order_date','delivery_date','user_dob','user_reg_date']:
    df2[col] = df2[col].dt.strftime("%d-%m-%Y")
```

In [6]:
```
df=df1.append(df2, ignore_index = True) # appnding the datasets
```

PRE-PROCESSING

In [7]:
```
#Converting to date-type
for col in ['order_date','delivery_date','user_dob','user_reg_date']:
    df[col] = pd.to_datetime(df[col], infer_datetime_format=True)
```

In [8]:
```
df['Delivery_days']= np.trunc((df['delivery_date'] - df['order_date'])/
np.timedelta64(1,'D'))#Crating column from existing ones
```

In [9]:
```
df['User_Age']= np.trunc((df['order_date'] - df['user_dob'])/np.timedel
ta64(1,'Y'))#Crating column from existing ones
```

In [10]:
```
df.head()
```

Out[10]:

	order_item_id	order_date	delivery_date	item_id	item_size	item_color	brand_id	iten
0	1	2016-06-22	2016-06-27	643	38	navy	30	49.9
1	10	2016-06-22	2016-06-27	195	xxl	grey	46	19.9
2	11	2016-06-22	2016-07-05	25	xxl	grey	5	79.9
3	32	2016-06-23	2016-06-26	173	m	brown	20	19.9
4	43	2016-06-23	2016-06-26	394	40	black	44	90.0

In [11]:
```
#VALIDATION CHECKS
df = df.dropna(how='any', subset=['delivery_date'])
df=df[df['Delivery_days']>=0]
```

In [13]:
```
df_train=df[df['train_test_indicator']=="train"]
```

```
df_test=df[df['train_test_indicator']=="test"]
```

In [14]:
```
print("train",df_train.shape)
print("test",df_test.shape)
```

```
train (71663, 17)
test (17947, 17)
```

In [15]:
```
df_test=df_test[df_test['item_id'].isin(df_train['item_id'].unique())]# item_ids selected in test data those are present in train
```

In [16]:
```
df_test.shape
```

Out[16]:
```
(17895, 17)
```

OUTLIER TREATMENT

In [17]:
```
print(min(df_train['User_Age']))
print(max(df_train['User_Age']))
```

```
4.0
115.0
```

In [18]:
```
#Using mean+-3*std.dev method
LL=df_train['User_Age'].mean() - 3*df_train['User_Age'].std()
UL=df_train['User_Age'].mean() + 3*df_train['User_Age'].std()
print("Lowest allowed",LL)
print("Highest allowed",UL)
((df_train['User_Age'] < LL) | (df_train['User_Age'] > UL)).sum()
```

```
Lowest allowed 18.1875060404912
Highest allowed 85.58820204050225
```

Out[18]:
```
927
```

In [19]:
```
#CAPPING
upper_limit = df_train['User_Age'].mean() + 3*df_train['User_Age'].std
()
```

```
lower_limit = df_train['User_Age'].mean() - 3*df_train['User_Age'].std
()

df_train['User_Age'] = np.where(df_train['User_Age']>upper_limit,upper_
limit,
                  np.where(df_train['User_Age']<lower_limit,lower_limit,
df_train['User_Age']))
```

In [20]:
```
print(min(df_train['User_Age']))
print(max(df_train['User_Age']))
```

```
18.1875060404912
85.58820204050225
```

In [21]:
```
#INTER QUARTILE RANGE METHOD

# Q1 = df_train['User_Age'].quantile(0.25)
# Q3 = df_train['User_Age'].quantile(0.75)
# IQR = Q3 - Q1
# LL= Q1 - 1.5 * IQR
# UL= Q3 + 1.5 * IQR
# print("Lowest allowed",LL)
# print("Highest allowed",UL)
# ((df_train['User_Age'] < LL) | (df_train['User_Age'] > UL)).sum()
# Lowest allowed 28.5
# Highest allowed 72.5
# 1833
```

In [22]:
```
#Outliers handled for test data in a similar fashion
upper_limit = df_test['User_Age'].mean() + 3*df_test['User_Age'].std()
lower_limit = df_test['User_Age'].mean() - 3*df_test['User_Age'].std()

df_test['User_Age'] = np.where(df_test['User_Age']>upper_limit,upper_li
mit,
                  np.where(df_test['User_Age']<lower_limit,lower_limit,
df_test['User_Age']))
```

HANDLING MISSING VALUES

In [23]:
```
df_train['User_Age'].isnull().sum()
```

Out[23]:
```
6275
```

In [24]:
```
df_train.groupby('user_title')['User_Age'].mean()
```

Out[24]:
```
user_title
Company         51.555556
Family          48.578014
Mr              51.595794
Mrs             51.501745
not reported    54.205882
Name: User_Age, dtype: float64
```

In [25]:
```
#Missing values handled through group means by a separate column (train
& test data separately)

df_train['User_Age'] = df_train['User_Age'].fillna(df_train.groupby('us
er_title')['User_Age'].transform('mean'))
df_train['User_Age'].isnull().sum()
```

Out[25]:
```
0
```

In [26]:
```
df_test['User_Age'] = df_test['User_Age'].fillna(df_test.groupby('user_
title')['User_Age'].transform('mean'))
df_test['User_Age'].isnull().sum()
```

Out[26]:
```
0
```

In [27]:
```
df_train['item_color'].unique()
```

Out[27]:
```
array(['navy', 'grey', 'brown', 'black', 'bordeaux', 'white', 'purple',
       'magenta', 'stained', 'blue', 'red', 'olive', 'ocher', 'ash',
       'khaki', 'nature', 'denim', 'curry', 'beige', 'turquoise', 'gree
n',
       'anthracite', 'yellow', 'berry', 'petrol', 'brwon', 'dark deni
m',
       'hibiscus', 'azure', 'ecru', 'gold', 'orange', 'silver',
```

```
       'darkblue', 'mocca', 'mint', 'pink', 'copper coin', 'jade', 'bla
u',
       'aqua', 'cobalt blue', 'mango', 'champagner', 'aubergine',
       'cognac', 'fuchsia', 'pallid', 'aquamarine', 'terracotta',
       'aviator', 'ancient', 'curled', 'apricot', 'coral', 'basalt',
       'floral', 'creme', 'mahagoni', 'dark garnet', 'striped', 'ivor
y',
       '?', 'dark oliv', 'dark navy', 'antique pink', 'habana',
       'dark grey', 'amethyst', 'currant purple', 'kanel', 'ebony',
       'avocado', 'caramel', 'baltic blue', 'almond', 'opal'],
      dtype=object)
```

In [28]:
```
df_train['item_color'].nunique()
```

Out[28]:
```
77
```

In [29]:
```
#Replace '?' categories by NaN
df_train['item_color']=df_train['item_color'].replace("?",np.NaN)
df_test['item_color']=df_test['item_color'].replace("?",np.NaN)
```

In [30]:
```
print("train_null",df_train['item_color'].isnull().sum())
print("test_null",df_test['item_color'].isnull().sum())
```

```
train_null 47
test_null 11
```

In [31]:
```
#Missing values handled through maximum frequency/mode
df_train['item_color'].fillna(df_train['item_color'].mode()[0], inplace
=True)
df_test['item_color'].fillna(df_test['item_color'].mode()[0], inplace=T
rue)
```

In [32]:
```
df_train['item_color'].value_counts()
```

Out[32]:
```
black        13528
blue          7590
brown         6338
grey          6256
```

```
red           5111
             ...
caramel          4
ebony            4
opal             2
creme            2
amethyst         2
Name: item_color, Length: 76, dtype: int64
```

CONVERTING CATEGORICAL TO CONTINUOUS VIA "TARGET MEAN ENCODING" TECHNIQUE

In [33]:
```
df_test["item_size"]= df_test["item_size"].astype(str)
```

In [34]:
```
for enc_vars in ['item_id','item_size','item_color','brand_id']:
    mean_encode=df_train.groupby(enc_vars)['return'].mean()
    df_train.loc[:,enc_vars+"_enc"]=df_train[enc_vars].map(mean_encode)
    df_test.loc[:,enc_vars+"_enc"]=df_test[enc_vars].map(mean_encode)
    print("train_null_"+enc_vars,df_train[enc_vars].isnull().sum())
    print("test_null_"+enc_vars,df_test[enc_vars].isnull().sum())
```

```
train_null_item_id 0
test_null_item_id 0
train_null_item_size 0
test_null_item_size 0
train_null_item_color 0
test_null_item_color 0
train_null_brand_id 0
test_null_brand_id 0
```

In [35]:
```
print(df_train['user_title'].value_counts())
print(df_test['user_title'].value_counts())
```

```
Mrs             68488
Mr               2729
Family            264
Company           104
not reported       78
```

```
Name: user_title, dtype: int64
Mrs             17130
Mr                668
Family             57
Company            23
not reported       17
Name: user_title, dtype: int64
```

CREATING DUMMIES FOR CATEGORICAL VARIABLES

In [36]:
```
df_train.groupby(['return','user_title']).size().unstack(fill_value=0)
#Chcking 'user_title' categorical variable
```

Out[36]:

user_title	Company	Family	Mr	Mrs	not reported
return					
0.0	40	141	1418	33672	47
1.0	64	123	1311	34816	31

In [37]:
```
df_train['return'].value_counts() # Event rate
```

Out[37]:
```
1.0    36345
0.0    35318
Name: return, dtype: int64
```

In [38]:
```
#Creating dummies for train & test data
df_train_state_dummies=pd.get_dummies(df_train['user_state'],drop_first
=True)
df_test_state_dummies=pd.get_dummies(df_test['user_state'],drop_first=T
rue)
```

In [39]:
```
df_train_user_dummies=pd.get_dummies(df_train['user_title'],drop_first=
True)
df_test_user_dummies=pd.get_dummies(df_test['user_title'],drop_first=Tr
ue)
```

CREATING A CATEGORICAL VARIABLE FROM CONTINUOUS THROUGH BINNING

In [40]:
```
#Creating Customer_Tenure variable & then binning it to form Customer_C
ategory in both train & test data

df_train["Customer_Tenure"]=np.trunc((df_train['order_date'] - df_train
['user_reg_date'])/np.timedelta64(1,'D'))
df_test["Customer_Tenure"]=np.trunc((df_test['order_date'] - df_test['u
ser_reg_date'])/np.timedelta64(1,'D'))
```

In [41]:
```
df_train["Customer_Category"]=pd.cut(df_train.Customer_Tenure,bins=[-5,
30,365,600],labels=['New_Customer','Slightly_Old_Customer','Old_Custome
r'])
```

In [42]:
```
df_train.groupby(['return','Customer_Category']).size().unstack(fill_va
lue=0)
```

Out[42]:

Customer_Category	New_Customer	Slightly_Old_Customer	Old_Customer
return			
0.0	9058	11687	14573
1.0	8961	11582	15802

In [43]:
```
dt=pd.crosstab(index=df_train['return'], columns=df_train['Customer_Cat
egory']) #Creating a contingency table
```

CHI-SQUARE TEST FOR 'Customer_Category' VARIABLE

In [44]:
```
#Checking null hypothesis is approved or rejected
from scipy.stats import chi2_contingency
data = dt
stat, p, dof, expected = chi2_contingency(data)
```

```
# interpret p-value
alpha = 0.05
print("p value is " + str(p))
if p <= alpha:
    print('Dependent (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
p value is 1.5139447020139232e-08
Dependent (reject H0)
```

In [45]:

```
df_test["Customer_Category"]=pd.cut(df_test.Customer_Tenure,bins=[-5,30
,365,600],labels=['New_Customer','Slightly_Old_Customer','Old_Customer'
])
```

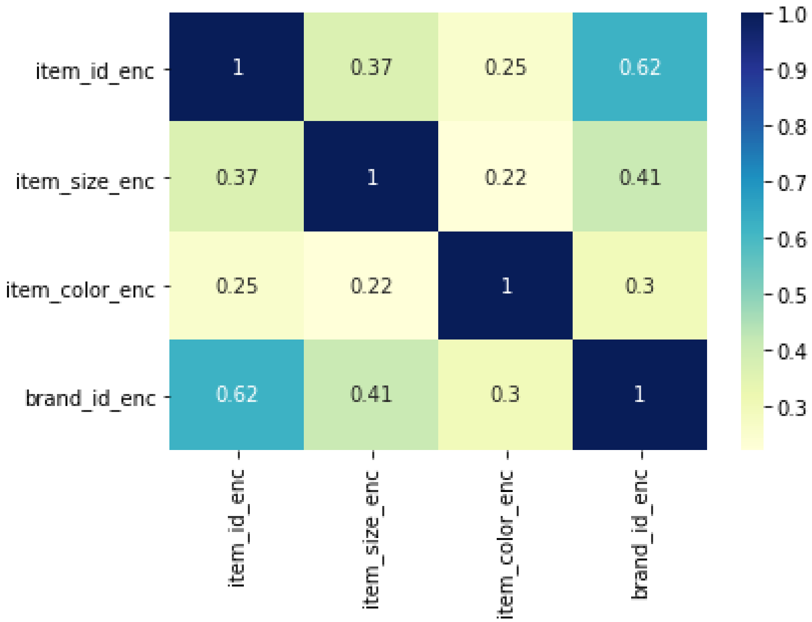
In [46]:

```
#Creating dummies for the same variable
df_train_cust_dummies=pd.get_dummies(df_train['Customer_Category'],drop
_first=True)
df_test_cust_dummies=pd.get_dummies(df_test['Customer_Category'],drop_f
irst=True)
```

In [47]:

```
#Checking the correlatin matrix
import seaborn as sns
sns.heatmap(df_train[['item_id_enc','item_size_enc','item_color_enc','b
rand_id_enc']].corr(),cmap="YlGnBu", annot=True)
```

Out[47]: <AxesSubplot:>



In [48]:
```
#CREATING BACK-UPS

df_train_bkp=df_train.copy()
df_test_bkp=df_test.copy()
```

In [49]:
```
df_train=df_train_bkp.copy()
df_test=df_test_bkp.copy()
```

In [51]:
```
#Concatenating continuous & dummies to form the final data sets
df_train=pd.concat([df_train[['Delivery_days','item_id_enc','item_size_
enc','item_color_enc','brand_id_enc',
                             'item_price','User_Age','return']],
                              df_train_user_dummies,df_train_state_dumm
ies,df_train_cust_dummies],axis=1)
df_test=pd.concat([df_test[['order_item_id','Delivery_days','item_id_en
c','item_size_enc','item_color_enc','brand_id_enc',
                             'item_price','User_Age','return']],
```

```
                              df_test_user_dummies,df_test_state_dummie
s,df_test_cust_dummies],axis=1)
```

In [140]:
```
# df_train=pd.concat([df_train[['Delivery_days','item_id_enc','item_siz
e_enc','item_color_enc','brand_id_enc',
#                               'item_price','User_Age','return']],
#                                df_train_user_dummies,df_train_state_du
mmies,df_train_cust_dummies],axis=1)
# df_test=pd.concat([df_test[['Delivery_days','item_id_enc','item_size_
enc','item_color_enc','brand_id_enc',
#                               'item_price','User_Age','return']],
#                                df_test_user_dummies,df_test_state_dumm
ies,df_test_cust_dummies],axis=1)
```

In [52]:
```
df_train.head()
```

Out[52]:

	Delivery_days	item_id_enc	item_size_enc	item_color_enc	brand_id_enc	item_price
0	5.0	0.235294	0.522085	0.368932	0.398649	49.9
1	5.0	0.418182	0.470209	0.531490	0.514104	19.9
2	13.0	0.484848	0.470209	0.531490	0.468740	79.9
3	3.0	0.698885	0.493287	0.536289	0.622444	19.9
4	3.0	0.627451	0.555998	0.514710	0.591647	90.0

5 rows × 29 columns

In [53]:
```
df_test.head()
```

Out[53]:

	order_item_id	Delivery_days	item_id_enc	item_size_enc	item_color_enc	brand_
79945	26	3.0	0.333333	0.496260	0.497696	0.4891
79946	28	9.0	0.469388	0.470209	0.475862	0.4580

	order_item_id	Delivery_days	item_id_enc	item_size_enc	item_color_enc	brand_
79947	37	3.0	0.584821	0.522085	0.487022	0.5486
79950	80	3.0	0.323944	0.492143	0.470396	0.3972
79951	95	3.0	0.323944	0.496260	0.507723	0.3972

5 rows × 30 columns

MODELING USING DIFFERENT ALGORITHMS

In [ ]:
```
#Selecting dependent & independent variables
df_train_X=df_train.drop('return',axis=1)
df_train_Y=df_train['return']
```

In [73]:
```
from sklearn.model_selection import train_test_split
```

In [74]:
```
#Train-Validation split
X_train,X_val,Y_train,Y_val=train_test_split(df_train_X,df_train_Y,test
_size=0.3,random_state=101)
```

In [75]:
```
print(X_train.shape)
print(Y_train.shape)
print(X_val.shape)
print(Y_val.shape)
```

```
(50164, 28)
(50164,)
(21499, 28)
(21499,)
```

DECISION TREE

In [114]:
```
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import GridSearchCV
```

In [149]:
```
dtree=DecisionTreeClassifier()
```

In [79]:
```
n_folds=5 #folds for Cross-validation
```

In [151]:
```
#Creating param-grid from selecting best set of hyper-parameters from t
he param-grid space
param_grid={'max_depth': range(5,15,5),
           'criterion': ['entropy','gini']}
```

In [152]:
```
grid_search=GridSearchCV(estimator=dtree,param_grid=param_grid,cv=n_fol
ds,verbose=1)#Using gridsearch CV
```

In [153]:
```
grid_search.fit(X_train,Y_train)#fitting the model
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    6.7s finished
```

Out[153]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['entropy', 'gini'],
                         'max_depth': range(5, 15, 5)},
             verbose=1)
```

In [156]:
```
cv_results=pd.DataFrame(grid_search.cv_results_)
```

In [157]:
```
cv_results #showing all possible results
```

Out[157]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param
0	0.252280	0.053752	0.016194	0.008100	entropy	5
1	0.425159	0.039539	0.011295	0.002308	entropy	10
2	0.224857	0.016101	0.013806	0.003363	gini	5
3	0.396868	0.024477	0.012980	0.000880	gini	10

In [166]:
```
#Selecting best heper-parameters
best_param_max_depth=grid_search.best_params_.get("max_depth")
best_param_criterion=grid_search.best_params_.get("criterion")
```

In [ ]:
```
#criterion: entropy
#max_depth: 5
```

In [170]:
```
#fitting the model with the optimal parameters
dtree_best=DecisionTreeClassifier(max_depth=best_param_max_depth,criter
ion=best_param_criterion)
dtree_best.fit(X_train,Y_train)
```

Out[170]: DecisionTreeClassifier(criterion='entropy', max_depth=5)

```
In [ ]: Y_pred=dtree_best.predict_proba()#predicting
```

```
In [61]: from sklearn.metrics import roc_auc_score
```

```
In [173]: #Computing Validation-AUC score
Y_pred=dtree_best.predict_proba(X_val)[:,1]
roc_auc_score(Y_val,Y_pred)
```

```
Out[173]: 0.6702318213796128
```

RANDOM FOREST

```
In [ ]: from sklearn.ensemble import RandomForestClassifier #Import RF library
```

```
In [176]: rf=RandomForestClassifier()
```

```
In [182]: #Providing hyper-parameter space through param-grid
param_grid={"max_depth": range(5,15,5),
            "n_estimators": [100,200,300,400,500],
            "max_features": [5,10,15,20,25],
            "criterion": ["entropy","gini"]}
```

```
In [183]: grid_search_rf= GridSearchCV(estimator=rf,param_grid=param_grid,cv=n_fo
lds,verbose=1,scoring="roc_auc")
```

```
In [184]: grid_search_rf.fit(X_train,Y_train)#fitting the model through grid-sear
ch
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed: 254.2min finished
```

```
Out[184]: GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['entropy', 'gini'],
                         'max_depth': range(5, 15, 5),
```

```
                          'max_features': [5, 10, 15, 20, 25],
                          'n_estimators': [100, 200, 300, 400, 500]},
             scoring='roc_auc', verbose=1)
```

In [185]:
```
cv_results_rf=pd.DataFrame(grid_search_rf.cv_results_)
cv_results_rf
```

Out[185]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	para
0	3.292910	0.131676	0.167140	0.005537	entropy	5
1	6.461277	0.190989	0.333885	0.046683	entropy	5
2	9.413540	0.057628	0.456435	0.008330	entropy	5
3	12.246904	0.053706	0.592449	0.008673	entropy	5
4	15.602071	0.089389	0.746025	0.013125	entropy	5
...	...	...	...	...	...	...

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	parai
95	16.694290	0.041676	0.241922	0.001701	gini	10
96	35.132709	3.023341	0.493758	0.039465	gini	10
97	51.212970	1.580567	0.722784	0.041189	gini	10
98	41.686685	13.612231	0.542948	0.194347	gini	10
99	45.945740	0.737254	0.605086	0.010985	gini	10

100 rows × 17 columns

In [191]:
```
#Selecting optimal parametrs
```

```
best_param_max_depth_rf=grid_search_rf.best_params_.get("max_depth")
best_param_n_estimators_rf=grid_search_rf.best_params_.get("n_estimator
s")
best_param_max_features_rf=grid_search_rf.best_params_.get("max_feature
s")
best_param_criterion_rf=grid_search_rf.best_params_.get("criterion")
```

In [195]:
```
best_param_n_estimators_rf
```

Out[195]:
```
400
```

In [ ]:
```
#max_depth: 10
#n_estmators: 400
#max_featues: 10
#criterion: entropy
```

In [196]:
```
#Fitting the model with optimal parameters
rf_best=RandomForestClassifier(max_depth=best_param_max_depth_rf,n_esti
mators=best_param_n_estimators_rf,
                              max_features=best_param_max_features_rf,c
riterion=best_param_criterion_rf)
rf_best.fit(X_train,Y_train)
```

Out[196]:
```
RandomForestClassifier(criterion='entropy', max_depth=10, max_features=
10,
                       n_estimators=400)
```

In [197]:
```
#Computing Validation-AUC score
Y_pred_rf=rf_best.predict_proba(X_val)[:,1]
roc_auc_score(Y_val,Y_pred_rf)
```

Out[197]:
```
0.6764268511677428
```

LOGISTIC REGRESSION

In [76]:
```
from sklearn.linear_model import LogisticRegression
```

In [77]:
```
lr=LogisticRegression()
```

In [80]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(lr, X_train, Y_train, scoring='roc_auc', cv=n_
folds, n_jobs=-1)
```

In [81]:
```
scores
```

Out[81]:
```
array([0.68966435, 0.68795443, 0.68968873, 0.67733043, 0.67703639])
```

In [82]:
```
sum(scores)/len(scores)
```

Out[82]:
```
0.6843348654723348
```

In [92]:
```
lr.fit(X_train, Y_train) #Fitting the model
```

Out[92]:
```
LogisticRegression()
```

In [103]:
```
Y_pred_lr=lr.predict_proba(X_val)[:,1] #predicting on validation data
```

In [105]:
```
#Computing Validation-AUC score
roc_auc_score(Y_val,Y_pred_lr)
```

Out[105]:
```
0.6737276606340082
```

XG-BOOST

In [58]:
```
#Importing libraries for XG-Boost
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
```

In [59]:
```
import matplotlib.pylab as plt
%matplotlib inline
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 12, 4
```

```
In [54]: target = 'return'
         #IDcol = 'item_id'
```

```
In [60]: from sklearn import metrics
```

```
In [55]: #Creating separate splits of train & validation data
         msk = np.random.rand(len(df_train)) < 0.8
         train = df_train[msk]
         val  = df_train[~msk]
```

```
In [67]: #Defining a function for XG-Boost fitting using cross-validation & stop
         ping_rounds;
         #also returning train-validation Accuracy-AUC scores
         def modelfit(alg, dtrain, predictors,useTrainCV=True, cv_folds=5, early
         _stopping_rounds=100):

             if useTrainCV:
                 xgb_param = alg.get_xgb_params()
                 xgtrain = xgb.DMatrix(dtrain[predictors].values, label=dtrain[t
         arget].values)
                 cvresult = xgb.cv(xgb_param, xgtrain, num_boost_round=alg.get_p
         arams()['n_estimators'], nfold=cv_folds,
                     metrics='auc', early_stopping_rounds=early_stopping_rounds)
                 alg.set_params(n_estimators=cvresult.shape[0])

             #Fit the algorithm on the data
             alg.fit(dtrain[predictors], dtrain['return'],eval_metric='auc')

             #Predict training set:
             dtrain_predictions = alg.predict(dtrain[predictors])
             dtrain_predprob = alg.predict_proba(dtrain[predictors])[:,1]


             #Print model report:
             print ("\nModel Report")
             print ("Accuracy_train : %.4g" % metrics.accuracy_score(dtrain['ret
         urn'].values, dtrain_predictions))
             print ("AUC Score (Train): %f" % metrics.roc_auc_score(dtrain['retu
```

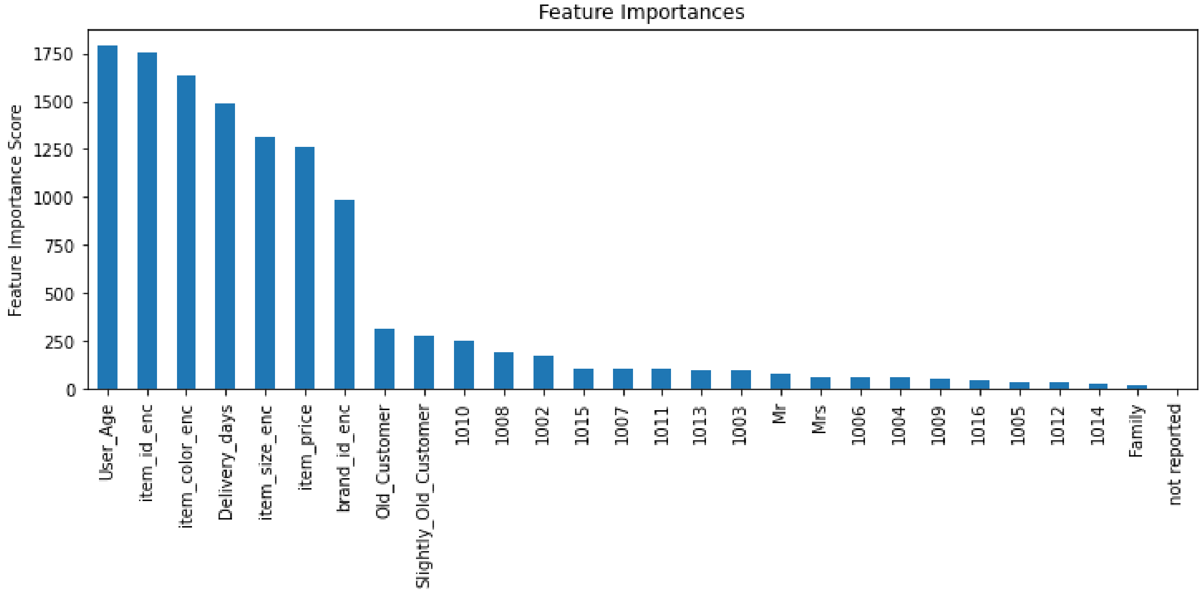
```
rn'], dtrain_predprob))
    dval_predictions = alg.predict(val[predictors])
    dval_predprob = alg.predict_proba(val[predictors])[:,1]
    print ("Accuracy_val : %.4g" % metrics.accuracy_score(val['return']
.values, dval_predictions))
    print ("AUC Score (val): %f" % metrics.roc_auc_score(val['return'],
dval_predprob))
    print(alg.set_params(n_estimators=cvresult.shape[0]))


    feat_imp = pd.Series(alg.get_booster().get_fscore()).sort_values(as
cending=False)
    feat_imp.plot(kind='bar', title='Feature Importances')
    plt.ylabel('Feature Importance Score')
```

In [68]:

```
#passing parameters to XGB (1st step: identifying the number of trees/n
_estimators with a high learning rate)
predictors = [x for x in train.columns if x not in [target]]#Set of ind
ependent variables
xgb1 = XGBClassifier(
 learning_rate =0.1,
 n_estimators=500,
 max_depth=10,
 min_child_weight=1,
 gamma=0,
 subsample=0.8,
 colsample_bytree=0.8,
#  objective= 'binary:logistic',
 nthread=-1,
 scale_pos_weight=1,
 seed=27)
modelfit(xgb1, train, predictors)
```

```
Model Report
Accuracy_train : 0.6971
AUC Score (Train): 0.778982
Accuracy_val : 0.6291
AUC Score (val): 0.678239
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
```

```
                 colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id
=-1,
                 importance_type='gain', interaction_constraints='',
                 learning_rate=0.1, max_delta_step=0, max_depth=10,
                 min_child_weight=1, missing=nan, monotone_constraints
='()',
                 n_estimators=32, n_jobs=8, nthread=-1, num_parallel_tree=
1,
                 random_state=27, reg_alpha=0, reg_lambda=1, scale_pos_wei
ght=1,
                 seed=27, subsample=0.8, tree_method='exact',
                 validate_parameters=1, verbosity=None)
```



In [249]:
```
#n_estimators=32
```

In [115]:
```
#Parameter tuning of tree_features by setting the optimal value of n_es
tiamaters
param_test1 = {
 'max_depth':range(3,15,2),
 'min_child_weight':range(1,6,2)
```

```
}
gsearch1 = GridSearchCV(estimator = XGBClassifier(learning_rate =0.1, n
_estimators=32,
 gamma=0, subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1, seed=27),
 param_grid = param_test1, scoring='roc_auc',n_jobs=4,iid=False, cv=5)
gsearch1.fit(df_train[predictors],df_train[target])
# gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
```

```
[02:35:02] WARNING: C:/Users/Administrator/workspace/xgboost-win64_rele
ase_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default e
valuation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
```

Out[115]:

```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8, gamma=0, gpu
_id=None,
                                     importance_type='gain',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step=
None,
                                     max_depth=None, min_child_weight=N
one,
                                     missing=nan, monotone_constraints=
None,
                                     n_estimators=32, n_jobs=None, nthr
ead=4,
                                     num_parallel_tree=None, random_sta
te=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=1, seed=27, subsa
mple=0.8,
                                     tree_method=None, validate_paramet
ers=None,
                                     verbosity=None),
             iid=False, n_jobs=4,
             param_grid={'max_depth': range(3, 15, 2),
```

```
param_grid={'max_depth':range(3, 10, 2),
                        'min_child_weight': range(1, 6, 2)},
scoring='roc_auc')
```

In [116]:

```
cv_results_xgb=pd.DataFrame(gsearch1.cv_results_)
cv_results_xgb
```

Out[116]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	pa
0	1.544093	0.063434	0.028935	0.004536	3	1
1	1.440733	0.032700	0.028550	0.003038	3	3
2	1.474021	0.033053	0.030828	0.001629	3	5
3	2.784468	0.058591	0.031855	0.006988	5	1
4	2.610554	0.077449	0.033537	0.004104	5	3
5	2.675755	0.068798	0.036893	0.005122	5	5
6	4.003752	0.044718	0.035222	0.004369	7	1

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	pa
7	3.922217	0.084039	0.033794	0.001143	7	3
8	3.978847	0.072159	0.034752	0.003862	7	5
9	5.570499	0.073526	0.038646	0.004617	9	1
10	5.335571	0.057993	0.038963	0.001960	9	3
11	5.219919	0.019704	0.037033	0.002522	9	5
12	8.088787	0.179468	0.042667	0.002910	11	1
13	7.252153	0.132804	0.041012	0.003173	11	3
14	6.800199	0.077353	0.044533	0.004913	11	5
15	9.663919	0.184463	0.050345	0.004330	13	1

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	pa
16	8.752636	0.109124	0.045236	0.001459	13	3
17	7.540864	0.834024	0.052174	0.005796	13	5

In [117]:
```
#Selecting best tree_fetures
best_param_max_depth_xgb=gsearch1.best_params_.get("max_depth")
best_param_min_child_weight_xgb=gsearch1.best_params_.get("min_child_we
ight")
```

In [119]:
```
best_param_min_child_weight_xgb
```

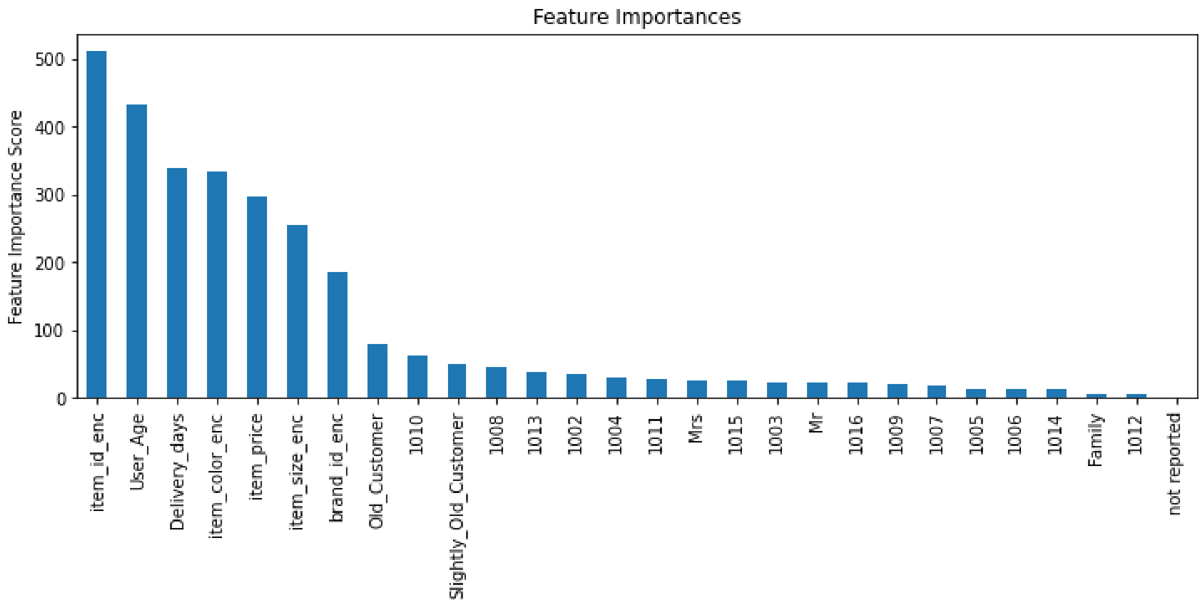
Out[119]: 1

In [260]:
```
#max_depth: 7
#min_child_weight: 1
```

In [120]:
```
#passing optimal n_estimators & tree_features to check train-validation
Accuracy & AUC
xgb1 = XGBClassifier(
 learning_rate =0.1,
 n_estimators=32,
 max_depth=7,
 min_child_weight=1,
 gamma=0,
 subsample=0.8,
 colsample_bytree=0.8,
#  objective= 'binary:logistic',
 nthread=-1,
 scale_pos_weight=1,
```

```
 seed=27)
modelfit(xgb1, train, predictors)
```

```
Model Report
Accuracy_train : 0.6502
AUC Score (Train): 0.713319
Accuracy_val : 0.63
AUC Score (val): 0.677659
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=0, gpu_id
=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints
='()',
              n_estimators=32, n_jobs=8, nthread=-1, num_parallel_tree=
1,
              random_state=27, reg_alpha=0, reg_lambda=1, scale_pos_wei
ght=1,
              seed=27, subsample=0.8, tree_method='exact',
              validate_parameters=1, verbosity=None)
```



In [121]:
```
#hyper-parameter tuning for gamma (which is used for x-treme pruning of trees)
param_test1 = {
 'gamma':range(0,5,1),
}
gsearch1 = GridSearchCV(estimator = XGBClassifier(learning_rate =0.1, n_estimators=32, max_depth=7,
 min_child_weight=1,
 subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=-1, scale_pos_weight=1, seed=27),
 param_grid = param_test1, scoring='roc_auc',n_jobs=4,iid=False, cv=5)
gsearch1.fit(df_train[predictors],df_train[target])
# gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
```

```
[02:39:08] WARNING: C:/Users/Administrator/workspace/xgboost-win64_rele
ase_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default e
valuation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
```

Out[121]:
```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8, gamma=None,
                                     gpu_id=None, importance_type='gai
n',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step=
None,
                                     max_depth=7, min_child_weight=1,
                                     missing=nan, monotone_constraints=
None,
                                     n_estimators=32, n_jobs=None, nthr
ead=-1,
                                     num_parallel_tree=None, random_sta
te=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=1, seed=27, subsa
```

```
mple=0.8,
                                        tree_method=None, validate_paramet
ers=None,
                                        verbosity=None),
             iid=False, n_jobs=4, param_grid={'gamma': range(0, 5)},
             scoring='roc_auc')
```

In [122]:
```
best_param_gamma_xgb=gsearch1.best_params_.get("gamma")#selecting best
 gamma
```

In [123]:
```
best_param_gamma_xgb
```

Out[123]:
```
2
```

In [ ]:
```
#gamma: 2
```

In [124]:
```
#Tuning for subsample & colsample by fixing rest of the hyper-parameter
s already obtained
param_test1 = {
  'subsample':[i/10.0 for i in range(6,10)],
  'colsample_bytree':[i/10.0 for i in range(6,10)]
}
gsearch1 = GridSearchCV(estimator = XGBClassifier(learning_rate =0.1, n
_estimators=32, max_depth=7,
 min_child_weight=1, gamma=2,
 objective= 'binary:logistic', nthread=-1, scale_pos_weight=1, seed=27
),
 param_grid = param_test1, scoring='roc_auc',n_jobs=4,iid=False, cv=5)
gsearch1.fit(df_train[predictors],df_train[target])
# gsearch1.grid_scores_, gsearch1.best_params_, gsearch1.best_score_
```

```
[02:42:03] WARNING: C:/Users/Administrator/workspace/xgboost-win64_rele
ase_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default e
valuation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
```

Out[124]:
```
GridSearchCV(cv=5,
```

```
             estimator=XGBClassifier(base_score=None, booster=None,
                                     colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, gamma=2,
                                     gpu_id=None, importance_type='gai
n',
                                     interaction_constraints=None,
                                     learning_rate=0.1, max_delta_step=
None,
                                     max_depth=7, min_child_weight=1,
                                     missing=nan, monotone_constraints=
None,
                                     n_estimators=32, n_jobs=None, nthr
ead=-1,
                                     num_parallel_tree=None, random_sta
te=None,
                                     reg_alpha=None, reg_lambda=None,
                                     scale_pos_weight=1, seed=27,
                                     subsample=None, tree_method=None,
                                     validate_parameters=None, verbosit
y=None),
             iid=False, n_jobs=4,
             param_grid={'colsample_bytree': [0.6, 0.7, 0.8, 0.9],
                         'subsample': [0.6, 0.7, 0.8, 0.9]},
             scoring='roc_auc')
```

In [125]:
```
#Selecting best colsumple & subsample
best_param_subsample_xgb=gsearch1.best_params_.get("subsample")
best_param_colsample_bytree_xgb=gsearch1.best_params_.get("colsample_by
tree")
```

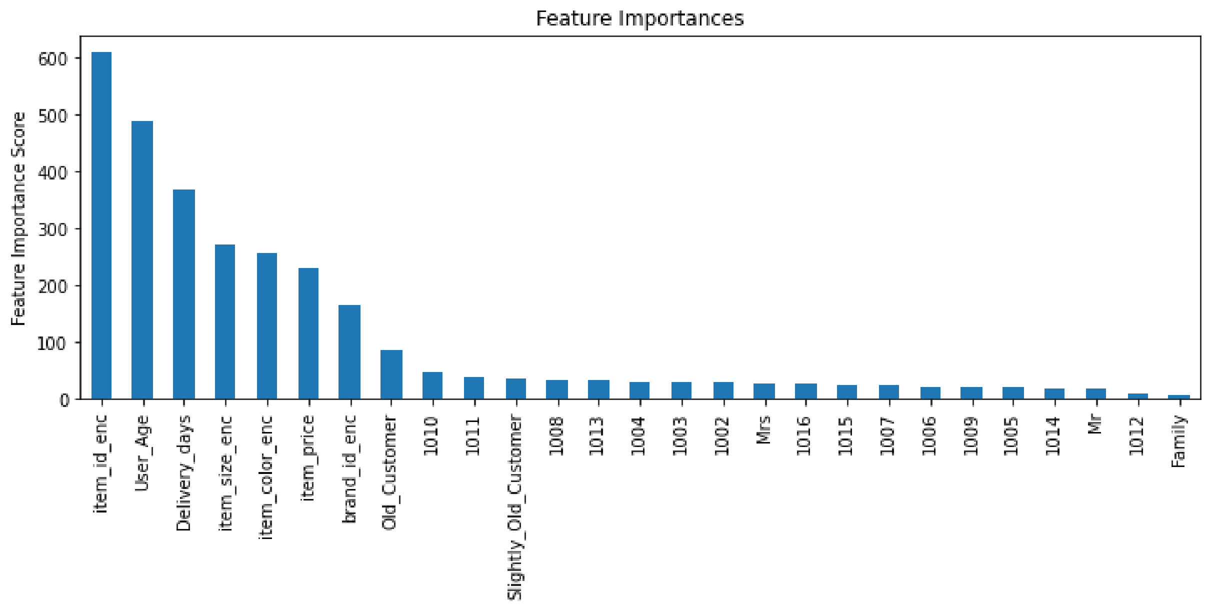
In [127]:
```
best_param_colsample_bytree_xgb
```

Out[127]:
```
0.8
```

In [ ]:
```
#subsample: 0.8
#colsample_bytree: 0.8
```

```
In [128]: #Setting all hyper-parameters to optimal-obtained values and fitting th
e model with a lower step-size/learning rate
xgb1 = XGBClassifier(
 learning_rate =0.01,
 n_estimators=32,
 max_depth=7,
 min_child_weight=1,
 gamma=2,
 subsample=0.8,
 colsample_bytree=0.8,
#  objective= 'binary:logistic',
 nthread=-1,
 scale_pos_weight=1,
 seed=27)
modelfit(xgb1, train, predictors)
```

```
Model Report
Accuracy_train : 0.6418
AUC Score (Train): 0.699715
Accuracy_val : 0.6262
AUC Score (val): 0.676687
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=2, gpu_id
=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.01, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints
='()',
              n_estimators=32, n_jobs=8, nthread=-1, num_parallel_tree=
1,
              random_state=27, reg_alpha=0, reg_lambda=1, scale_pos_wei
ght=1,
              seed=27, subsample=0.8, tree_method='exact',
              validate_parameters=1, verbosity=None)
```



FITTING THE MODEL ON THE ENTIRE TRAIN DATA SET WITH OPTIMAL HYPER-PARAMETERS

In [129]:

```
#Fitting the model with optimal parameters
xgb_best=XGBClassifier(subsample=0.8,scale_pos_weight=1,n_estimators=32
,min_child_weight=1,
                       max_depth=7,learning_rate=0.01,gamma=2,colsample
_bytree=0.8)
```

In [131]:

```
xgb_best.fit(df_train[predictors],df_train['return'])
```

```
[02:51:53] WARNING: C:/Users/Administrator/workspace/xgboost-win64_rele
ase_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default e
valuation metric used with the objective 'binary:logistic' was changed
```

```
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to
restore the old behavior.
```

Out[131]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=2, gpu_id
=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.01, max_delta_step=0, max_depth=7,
              min_child_weight=1, missing=nan, monotone_constraints
='()',
              n_estimators=32, n_jobs=8, num_parallel_tree=1, random_st
ate=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=
0.8,
              tree_method='exact', validate_parameters=1, verbosity=Non
e)
```

PREDICTING ON TRAIN AS WELL AS TEST DATA SET

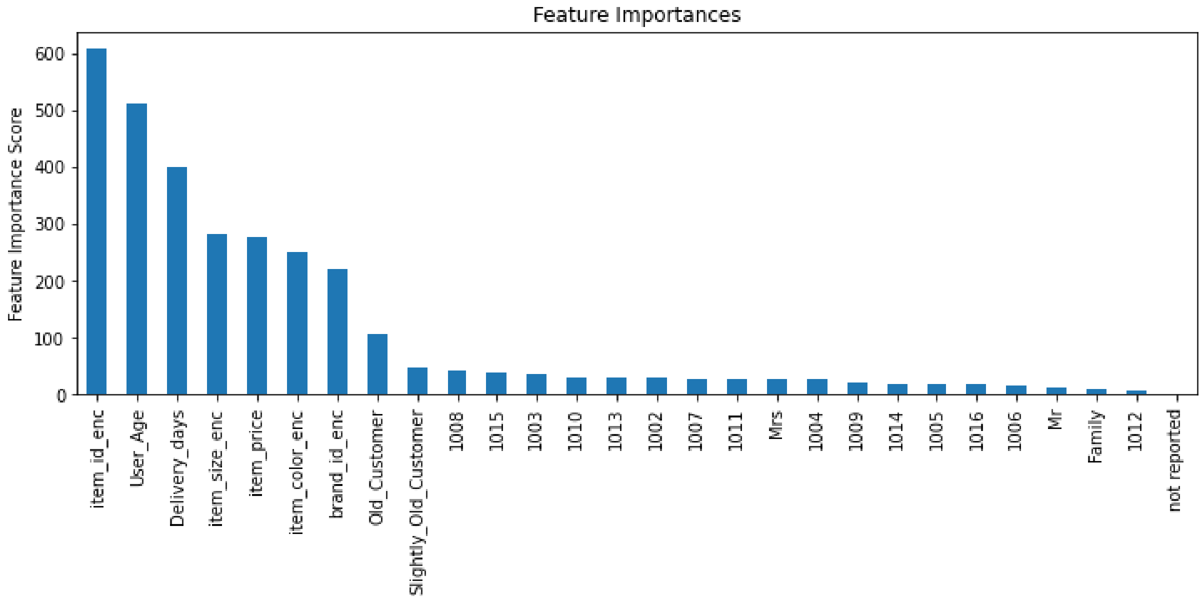
In [133]:

```
#Predict training set:
df_train_predictions = xgb_best.predict(df_train[predictors])
df_train_predprob = xgb_best.predict_proba(df_train[predictors])[:,1]


#Print model report:
print ("\nModel Report")
print ("Accuracy_train : %.4g" % metrics.accuracy_score(df_train['retur
n'].values, df_train_predictions))
print ("AUC Score (Train): %f" % metrics.roc_auc_score(df_train['retur
n'], df_train_predprob))

#Predicting on Test Data
df_test['Predicted'] = xgb_best.predict(df_test[predictors])
df_test['Predicted_Prob'] = xgb_best.predict_proba(df_test[predictors])
[:,1]

#Feature Importance
feat_imp = pd.Series(xgb_best.get_booster().get_fscore()).sort_values(a
scending=False)
```

```
feat_imp.plot(kind='bar', title='Feature Importances')
plt.ylabel('Feature Importance Score')
```

```
Model Report
Accuracy_train : 0.6397
AUC Score (Train): 0.696059
```

Out[133]: Text(0, 0.5, 'Feature Importance Score')



In [134]:
```
df_test.head()
```

Out[134]:

	order_item_id	Delivery_days	item_id_enc	item_size_enc	item_color_enc	brand_
79945	26	3.0	0.333333	0.496260	0.497696	0.4891
79946	28	9.0	0.469388	0.470209	0.475862	0.4580
79947	37	3.0	0.584821	0.522085	0.487022	0.5486
79950	80	3.0	0.323944	0.492143	0.470396	0.3972

	order_item_id	Delivery_days	item_id_enc	item_size_enc	item_color_enc	brand_
79951	95	3.0	0.323944	0.496260	0.507723	0.3972

5 rows × 32 columns

In [135]:
```
#Creating the final data set
df_final=df_test[['order_item_id','Predicted']]
```

In [136]:
```
delv_df=pd.merge(df2,df_final,on='order_item_id',how='left')#merge the
 predicted values to the test-data(initially read) set
```

In [139]:
```
delv_df.head()
```

Out[139]:

	order_item_id	order_date	delivery_date	item_id	item_size	item_color	brand_id	iten
0	26	23-06-2016	26-06-2016	92	xl	turquoise	42	69.9
1	28	23-06-2016	02-07-2016	2	xxl	green	2	19.9
2	37	23-06-2016	26-06-2016	895	38	white	39	39.9
3	56	23-06-2016	NaN	5	l	white	5	69.9
4	65	23-06-2016	NaN	55	40	purple	1	89.9

In [138]:
```
delv_df.drop("train_test_indicator",axis=1,inplace=True)#dropping train
_test indicator variable
```

Since some of the delivery dates are missing in the test data and some of them have delivery dates which are far earlier than order date, we assume that for those line items predicted values would be zero since there would be no chance of return

In [140]:
```
delv_df.to_csv("Predicted_Test_Data_SumeetRoutray.csv")#writing the fin
al output(consisting of predicted column) to csv
```

---------------------THE END--------------------