In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import train_test_split, KFold

from sklearn.metrics import confusion_matrix, roc_curve, auc, roc_auc_score, accuracy_s
core
from scipy.sparse import hstack


from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

import pickle
import tqdm

import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

from wordcloud import WordCloud, STOPWORDS

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

import xgboost as xgb

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

In [2]:

```
data = pd.read_csv('preprocessed_data.csv', nrows=50000)
```

In [3]:

```
data.shape
```

Out[3]:

```
(50000, 9)
```

Spliting X and Y

In [4]:

```
y = data.project_is_approved
X = data.drop(["project_is_approved"], axis=1)
```

Spliting Train, Test and Cross Validation datasets

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.33, stratify=y)
```

Defing one hot encoding functions

In [6]:

```
def convert_text_into_TFIDF(feature):
    '''This function transforms a feature into TFIDF vector'''

    vectorizer = TfidfVectorizer(min_df = 10, ngram_range=(1,4), max_features=5000)
    vectorizer.fit(X_train[feature].values)

    X_train_ohe = vectorizer.transform(X_train[feature].values)
    X_test_ohe = vectorizer.transform(X_test[feature].values)

    return vectorizer, X_train_ohe, X_test_ohe
```

In [7]:

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words = set(model.keys())

def calculate_TFIDF_Weighted_W2V(feature_value, dictionary, tfidf_words):


    tfidf_w2v_vectors = []

    for sentence in feature_value:
        vector = np.zeros(300)
        tf_idf_weight = 0

        for word in sentence.split():
            if(word in tfidf_words) and (word in glove_words):
                vec = model[word]

                tf_idf = dictionary[word] * (sentence.count(word)/len(sentence.split
()))
                vector += (vec * tf_idf)
                tf_idf_weight += tf_idf

        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors.append(vector)

    return tfidf_w2v_vectors

def convert_Text_Into_TFIDF_Weighted_W2V(feature):

    tfidf_model = TfidfVectorizer()
    tfidf_model.fit(X_train[feature].values)

    dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
    tfidf_words = set(tfidf_model.get_feature_names())

    X_train_tfidf_w2v = calculate_TFIDF_Weighted_W2V(X_train[feature].values, dictionar
y, tfidf_words)
    X_test_tfidf_w2v = calculate_TFIDF_Weighted_W2V(X_test[feature].values, dictionary,
tfidf_words)

    return tfidf_model, X_train_tfidf_w2v, X_test_tfidf_w2v
```

In [8]:

```
def get_response_code(X_train, y_train):

    df = pd.DataFrame({ 'X_train': X_train, 'y_train' : y_train })

    categories = df.X_train.unique()

    lst_response_code = []

    for category in categories:

        probablity_0 = 0
        probablity_1 = 0

        df_category = df[df.X_train == category]

        for i in range(len(df_category)):
            if df_category.X_train.values[i] == category:
                if df_category.y_train.values[i] == 0:
                    probablity_0 += 1
                else:
                    probablity_1 += 1

        response_code = {'category' : category,
                         'category_0' : probablity_0,
                         'category_1' : probablity_1,
                         'category_tot': probablity_0+ probablity_1 }

        lst_response_code.append(response_code)

    return pd.DataFrame(lst_response_code)
```

In [9]:

```
def encode_categorical_with_response_code(X, df_response_code):

    lst_category_0_1 = []
    for data in X:

        df_res = df_response_code[df_response_code.category == data]

        if len(df_res) != 0:
            lst_category_0_1.append([df_res.category_0.values[0] / df_res.category_tot.
values[0],
                                     df_res.category_1.values[0] / df_res.category_tot.va
lues[0]])

        else:
            lst_category_0_1.append([0.5, 0.5])

    return lst_category_0_1
```

In [10]:

```
def convert_categorical_Into_1_and_0(feature):

    df_response_code = get_response_code(X_train[feature], y_train)

    X_train_0_1 = encode_categorical_with_response_code(X_train[feature].values, df_res
ponse_code)
    X_test_0_1 = encode_categorical_with_response_code(X_test[feature].values, df_respo
nse_code)

    return df_response_code, X_train_0_1, X_test_0_1
```

In [11]:

```
def encode_numerical_features(feature):

    normalizer = Normalizer()
    normalizer.fit(X_train[feature].values.reshape(1,-1))

    X_train_ohe = normalizer.transform(X_train[feature].values.reshape(1,-1))
    X_test_ohe = normalizer.transform(X_test[feature].values.reshape(1,-1))

    return normalizer, X_train_ohe.reshape(-1,1), X_test_ohe.reshape(-1,1)
```

In [12]:

```
def get_essay_sentiment_score(X_essay):

    sid = SentimentIntensityAnalyzer()

    lst_sentiments = []

    for data in X_essay:
        ss = sid.polarity_scores(data)

        lst_sentiments.append([ss['neg'], ss['neu'], ss['pos'], ss['compound']])

    return lst_sentiments
```

In [13]:

```
'''
#Categorical
X_train['school_state']
X_train['teacher_prefix']
X_train['project_grade_category']
X_train['clean_categories']
X_train['clean_subcategories']

#numerical
X_train['teacher_number_of_previously_posted_projects']
X_train['price']

#essay
X_train['essay']'''
```

Out[13]:

```
"\n#Categorical\nX_train['school_state']\nX_train['teacher_prefix']\nX_tra
in['project_grade_category']\nX_train['clean_categories']\nX_train['clean_
subcategories']\n\n#numerical\nX_train['teacher_number_of_previously_poste
d_projects']\nX_train['price']\n\n#essay\nX_train['essay']"
```

Applying One hot encoding on categorical features

In [14]:

```
#Performing Response code encoding for School State
df_rspnse_code_state, X_train_state_0_1, X_test_state_0_1 \
                                    = convert_categorical_Into_1_and_0('school_state')
lst_state_features = df_rspnse_code_state.category.values


#Performing Response code encoding for tech_prefix
df_rspnse_code_tchr_prfx, X_train_tchr_prfx_0_1, X_test_tchr_prfx_0_1 \
                                    = convert_categorical_Into_1_and_0('teacher_prefix'
)
lst_tchr_prfx_features = df_rspnse_code_tchr_prfx.category.values

#Performing Response code encoding for grade_category
df_rspnse_code_grade, X_train_grade_0_1, X_test_grade_0_1 \
                                    = convert_categorical_Into_1_and_0('project_grade_c
ategory')
lst_grade_features = df_rspnse_code_grade.category.values

#Performing Response code encoding for categories
df_rspnse_code_categories, X_train_categories_0_1, X_test_categories_0_1 \
                                    = convert_categorical_Into_1_and_0('clean_categorie
s')
lst_categories_features = df_rspnse_code_categories.category.values

#Performing Response code encoding for subcategories
df_rspnse_code_subcategories, X_train_subcategories_0_1, X_test_subcategories_0_1 \
                                    = convert_categorical_Into_1_and_0('clean_subcatego
ries')
lst_subcategories_features = df_rspnse_code_subcategories.category.values
```

Applying Normalizing on numerical features

In [15]:

```
#Permoring normalization on teacher_number_of_previously_posted_projects
normalizer_prev_projects, X_train_prev_projects, X_test_prev_projects \
                        = encode_numerical_features('teacher_number_of_previously_poste
d_projects')

#Permoring normalization on price
normalizer_price, X_train_price, X_test_price = encode_numerical_features('price')
```

Essay Sentiment Score

In [16]:

```
lst_train_sentiments = get_essay_sentiment_score(X_train['essay'].values)

lst_test_sentiments = get_essay_sentiment_score(X_test['essay'].values)
```

Applying One hot encoding on textual features (TFIDF)

In [17]:

```
#Performing TFIDF feature vectoriztion on essay
tfdif_vectorizer_essay, X_train_essay_tfidf, X_test_essay_tfidf = convert_text_into_TFI
DF('essay')
```

Applying One hot encoding on textual features (TFIDF Word2Vec)

In [18]:

```
#Performing TFIDF word2vec feature vectoriztion on essay
w2v_vectorizer_essay, X_train_essay_w2v, X_test_essay_w2v = convert_Text_Into_TFIDF_Wei
ghted_W2V('essay')
```

Concatenating all features

In [19]:

```
#Concatinating feature set 1(TFIDF)
X_tr_set1 = hstack((X_train_state_0_1, X_train_tchr_prfx_0_1, X_train_grade_0_1, X_trai
n_categories_0_1,
                    X_train_subcategories_0_1, X_train_prev_projects, X_train_price, X_
train_essay_tfidf,
                    lst_train_sentiments)).tocsr()


X_te_set1 = hstack((X_test_state_0_1, X_test_tchr_prfx_0_1, X_test_grade_0_1, X_test_ca
tegories_0_1,
                    X_test_subcategories_0_1, X_test_prev_projects, X_test_price, X_tes
t_essay_tfidf,
                    lst_test_sentiments)).tocsr()
```

In [20]:

```
#Concatinating feature set 1(TFIDF Word2Vec)
X_tr_set2 = np.hstack((X_train_state_0_1, X_train_tchr_prfx_0_1, X_train_grade_0_1, X_t
rain_categories_0_1,
                    X_train_subcategories_0_1, X_train_prev_projects, X_train_price, X_
train_essay_w2v,
                    lst_train_sentiments))


X_te_set2 = np.hstack((X_test_state_0_1, X_test_tchr_prfx_0_1, X_test_grade_0_1, X_test
_categories_0_1,
                    X_test_subcategories_0_1, X_test_prev_projects, X_test_price, X_tes
t_essay_w2v,
                    lst_test_sentiments))
```

In [21]:

```
class CV_Results:
    def __init__(self, depth, estimator, acc_score):
        self.depth = depth
        self.estimator = estimator
        self.acc_score = acc_score
```

In [34]:

```
def getTrain_and_Cv_scores(grid_search):

    lst_train_scores = []
    lst_cv_scores = []


    idx_score = 0
    for i, depth in enumerate(depth_range):
        for j, estimator in enumerate(n_estimator):

            train_scores = []
            cv_scores = []

            for k in range(n_folds):
                k_fold_train_score = grid_search.cv_results_['split' + str(k) +'_train_
score'][idx_score]
                k_fold_cv_score = grid_search.cv_results_['split' + str(k) +'_test_scor
e'][idx_score]

                train_scores.append(k_fold_train_score)
                cv_scores.append(k_fold_cv_score)

            idx_score += 1

            train_result = CV_Results(depth, estimator, np.mean(np.array(train_scores
)))
            cv_result = CV_Results(depth, estimator, np.mean(np.array(cv_scores)))

            lst_train_scores.append(train_result)
            lst_cv_scores.append(cv_result)

    return lst_train_scores, lst_cv_scores
```

In [24]:

```
def print_train_cv_score(lst_train_scores, lst_cv_scores):
    print('\n---------Train scores----------')
    for scores in lst_train_scores:
        print(f'Depth :{scores.depth} Estimator :{scores.estimator} Auc score:{scores.a
cc_score}')


    print('\n---------CV scores----------')
    for scores in lst_cv_scores:
        print(f'Depth :{scores.depth} Estimator :{scores.estimator} Auc score:{scores.a
cc_score}')
```

In [25]:

```
def plot_HeatMap(lst_train_scores, lst_cv_scores):

    lst_train_cv_scores = [lst_train_scores, lst_cv_scores]

    fig = plt.figure(figsize = (15,5))

    plot_counter = 1
    for data in lst_train_cv_scores:

        estimator = []
        depth = []
        auc_scores = []

        for x in data:
            estimator.append(x.estimator)
            depth.append(x.depth)
            auc_scores.append(x.acc_score)


        #https://stackoverflow.com/questions/45470882/x-y-z-array-data-to-heatmap/45660
022
        df = pd.DataFrame.from_dict(np.array([estimator,depth,auc_scores]).T)
        df.columns = ['Estimator','Depth','AUC Scores']
        df['AUC Scores'] = pd.to_numeric(df['AUC Scores'])

        pivotted= df.pivot('Estimator','Depth','AUC Scores')

        sns.heatmap(pivotted, ax= fig.add_subplot(1, 2, plot_counter), annot=True, cmap
='coolwarm')

        if plot_counter == 1:
            plt.title('Train AUC Scores for each Hyperparameter')
        else:
            plt.title('CV AUC Scores for each Hyperparameter')

        plot_counter +=1
```

In [26]:

```
def plot_AUC(train_fpr, train_tpr, test_fpr, test_tpr, train_auc, test_auc, title):
    '''This function plot AUC curve for both train and test FPR and TPR'''

    plt.plot(train_fpr, train_tpr, label= f"Train AUC = {train_auc}" )
    plt.plot(test_fpr, test_tpr, label = f"Test AUC = {test_auc}")
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

    plt.legend()
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(title)

    plt.grid()
    plt.show()
```

In [27]:

```
def create_wordcloud_using_fp_essay(X_test_fp_essay):

    comment_words = ' '
    stopwords = set(STOPWORDS)

    # iterate through the csv file
    for val in X_test_fp_essay:

        # typecaste each val to string
        val = str(val)

        # split the value
        tokens = val.split()

        # Converts each token into lowercase
        for i in range(len(tokens)):
            tokens[i] = tokens[i].lower()

        for words in tokens:
            comment_words = comment_words + words + ' '


    wordcloud = WordCloud(width = 800, height = 800,
                    background_color ='white',
                    stopwords = stopwords,
                    min_font_size = 10).generate(comment_words)

    # plot the WordCloud image
    plt.figure(figsize = (8, 8), facecolor = None)
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.tight_layout(pad = 0)

    plt.show()
```

In [28]:

```
def plot_Box_plot(X_te_fp_price):
    sns.boxplot(data= X_te_fp_price)
    plt.xlabel("Price")
    plt.ylabel("Percentiles")
    plt.title("Price vs Likelihood of more datapoints")
    plt.show()
```

In [29]:

```
def plot_PDF(X_test_fp_tchr_prfx_0, X_test_fp_tchr_prfx_1):
    plt.close()

    a4_dims = (7, 7)
    fig, ax = plt.subplots(figsize=a4_dims)

    #https://python-graph-gallery.com/25-histogram-with-several-variables-seaborn/
    sns.distplot(X_test_fp_tchr_prfx_0, bins=10, color="skyblue", label="0 Response Cod
e", ax=ax)
    sns.distplot(X_test_fp_tchr_prfx_1, bins=10, color="red", label="1 Response Code",
ax=ax)
    plt.legend()

    plt.xlabel("Bins")
    plt.ylabel("Likelihood of false positive")
    plt.title("PDF plot")
    plt.grid()
    plt.show();
```

In [30]:

```
depth_range = [1, 2, 3, 4, 5]
n_estimator = [70, 80, 90, 100]


param = {'max_depth': depth_range, 'n_estimators': n_estimator}
n_folds = 3
```

Train model with Train set 1

In [31]:

```
clf = xgb.XGBClassifier()

#Finding best alpha using GridSearchCV method
clf_set1 = GridSearchCV(estimator = clf, param_grid= param, cv=n_folds, scoring='roc_au
c')
clf_set1.fit(X_tr_set1, y_train)

clf_set1.best_params_
```

Out[31]:

```
{'max_depth': 5, 'n_estimators': 100}
```

Getting Train and CV Scores for set 1

In [35]:

```
lst_train_scores_set1, lst_cv_scores_set1 = getTrain_and_Cv_scores(clf_set1)
```

Printing Train and CV Scores for set 1

In [36]:

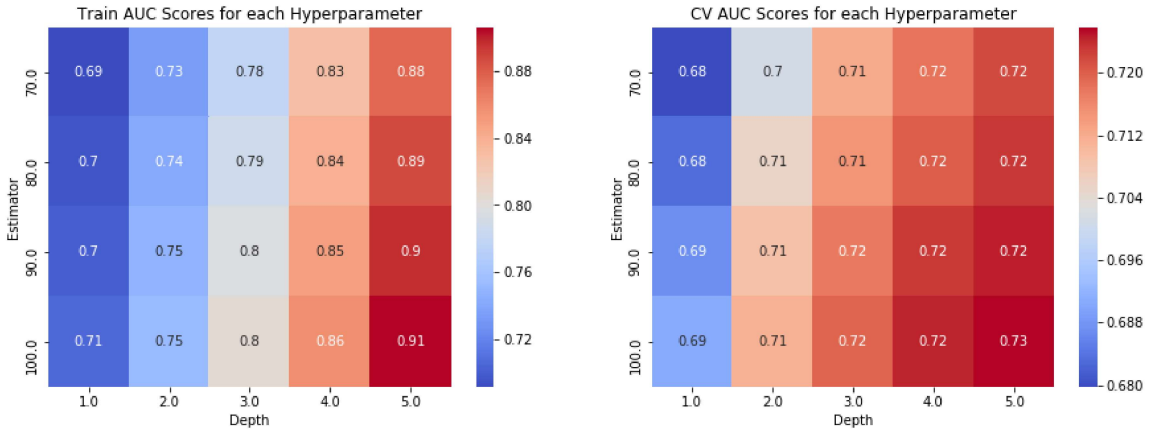
```
print_train_cv_score(lst_train_scores_set1, lst_cv_scores_set1)
```

```
---------Train scores----------
Depth :1 Estimator :70 Auc score:0.6917810881898792
Depth :1 Estimator :80 Auc score:0.6967986391951267
Depth :1 Estimator :90 Auc score:0.7011176212903741
Depth :1 Estimator :100 Auc score:0.7054608850304072
Depth :2 Estimator :70 Auc score:0.7344546017703893
Depth :2 Estimator :80 Auc score:0.7414195735482929
Depth :2 Estimator :90 Auc score:0.748092799033699
Depth :2 Estimator :100 Auc score:0.7534913218706113
Depth :3 Estimator :70 Auc score:0.7793973602632183
Depth :3 Estimator :80 Auc score:0.7882305119677396
Depth :3 Estimator :90 Auc score:0.7964637102264508
Depth :3 Estimator :100 Auc score:0.8043971925590699
Depth :4 Estimator :70 Auc score:0.828473356783869
Depth :4 Estimator :80 Auc score:0.8398540068765737
Depth :4 Estimator :90 Auc score:0.849024843332467
Depth :4 Estimator :100 Auc score:0.8578013496753987
Depth :5 Estimator :70 Auc score:0.8762801601751322
Depth :5 Estimator :80 Auc score:0.8878837398298897
Depth :5 Estimator :90 Auc score:0.8972952217584641
Depth :5 Estimator :100 Auc score:0.9060031459731762

---------CV scores----------
Depth :1 Estimator :70 Auc score:0.6797815775816379
Depth :1 Estimator :80 Auc score:0.6832641843538981
Depth :1 Estimator :90 Auc score:0.6869779361326266
Depth :1 Estimator :100 Auc score:0.6906033712335228
Depth :2 Estimator :70 Auc score:0.7009985715870778
Depth :2 Estimator :80 Auc score:0.7053517726162376
Depth :2 Estimator :90 Auc score:0.7089294261839378
Depth :2 Estimator :100 Auc score:0.7112601711805181
Depth :3 Estimator :70 Auc score:0.7121756444354203
Depth :3 Estimator :80 Auc score:0.7147380748360117
Depth :3 Estimator :90 Auc score:0.7173976710301236
Depth :3 Estimator :100 Auc score:0.7195528454727761
Depth :4 Estimator :70 Auc score:0.7181766162882891
Depth :4 Estimator :80 Auc score:0.7203525643587417
Depth :4 Estimator :90 Auc score:0.7230099770372579
Depth :4 Estimator :100 Auc score:0.7245353088781424
Depth :5 Estimator :70 Auc score:0.7216674617684485
Depth :5 Estimator :80 Auc score:0.7232416016846912
Depth :5 Estimator :90 Auc score:0.7245768951154535
Depth :5 Estimator :100 Auc score:0.7257989739658983
```

Plotting 3D graph for Train and CV Scores for set 1

In [37]:

```
plot_HeatMap(lst_train_scores_set1, lst_cv_scores_set1)
```



Training model with best parameter set 1

In [40]:

```
GBDT_clf_set1 = xgb.XGBClassifier(max_depth= clf_set1.best_params_['max_depth'],
                                                    n_estimators= clf_set1.best_params_['n_
estimators'])

GBDT_clf_set1.fit(X_tr_set1, y_train)
```

Out[40]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=5, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
```

Finding AUC curve on train and test data for set 1

In [41]:

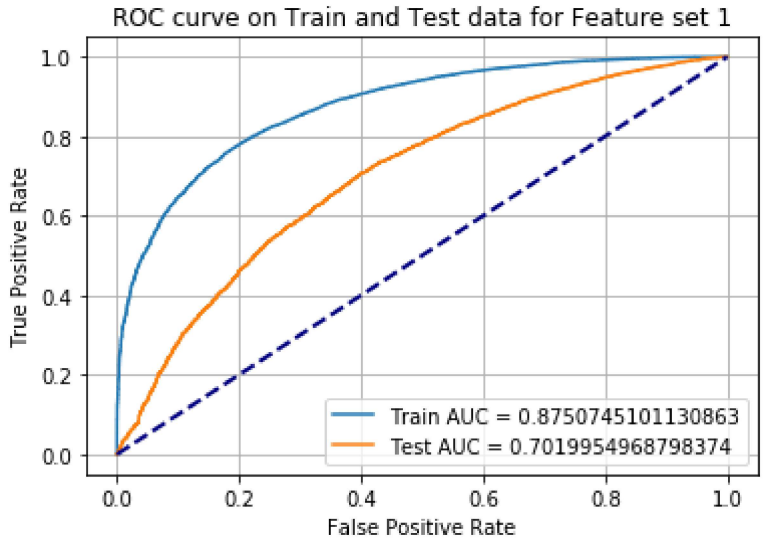
```
y_train_proba_set1 = GBDT_clf_set1.predict_proba(X_tr_set1)[:,1]
y_test_proba_set1 = GBDT_clf_set1.predict_proba(X_te_set1)[:,1]


#Finding AUC on train and test data
train_auc_set1 = roc_auc_score(y_train, y_train_proba_set1)
print('Train Auc for set 1')
print(train_auc_set1)

test_auc_set1 = roc_auc_score(y_test, y_test_proba_set1)
print('\n Test Auc for set 1')
print(test_auc_set1)
```

```
Train Auc for set 1
0.8750745101130863

 Test Auc for set 1
0.7019954968798374
```

Plotting ROC curve of train and test data for set 1

In [42]:

```
#Finding FPR and TPR both on train and test
train_fpr_set1, train_tpr_set1, train_threshold_set1 = roc_curve(y_train, y_train_proba
_set1)
test_fpr_set1, test_tpr_set1, test_threshold_set1 = roc_curve(y_test, y_test_proba_set1
)

#Plotting AUC curve
plot_AUC(train_fpr_set1, train_tpr_set1, test_fpr_set1, test_tpr_set1, train_auc_set1,
test_auc_set1,
         'ROC curve on Train and Test data for Feature set 1')
```



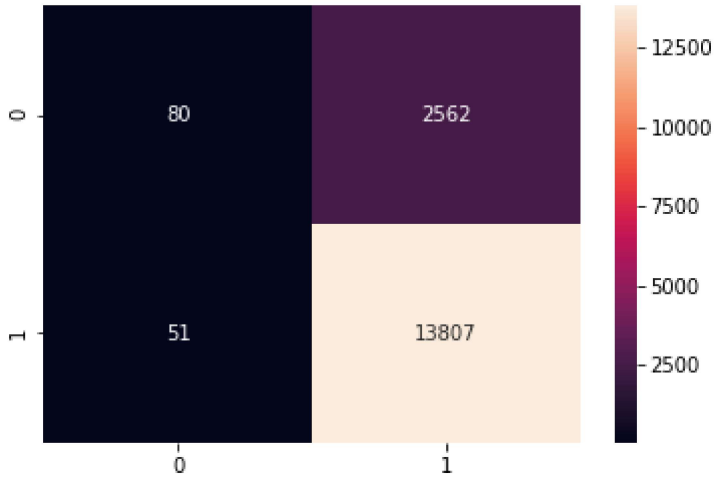
Plotting confusion matrix for set 1

In [43]:

```
#Predicting y_test
y_test_pred_set1 = GBDT_clf_set1.predict(X_te_set1)

confusion_matrix_set1 = confusion_matrix(y_test, y_test_pred_set1)

#Seaborn Heatmap representaion of Train confusion matrix
sns.heatmap(confusion_matrix_set1, annot=True, fmt="d")
```

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f71cd75ba8>
```



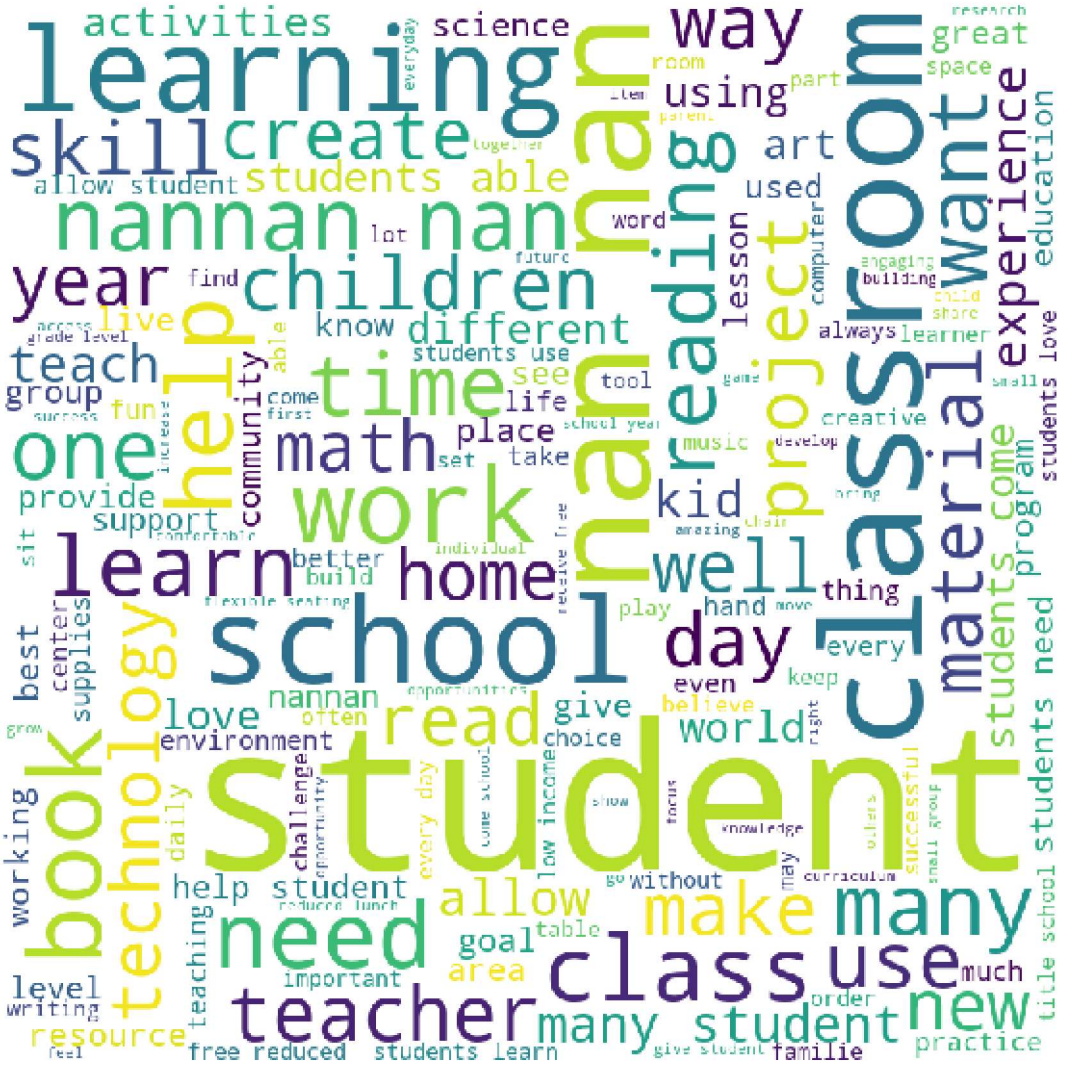
In [44]:

```
#Extracting False positive datapoints

fp_indices = []
for i in range(len(y_test)):
    if (np.array(y_test)[i] == 0) & (y_test_pred_set1[i] == 1):
        fp_indices.append(i)
```

Exploratory Data analysis for set 1

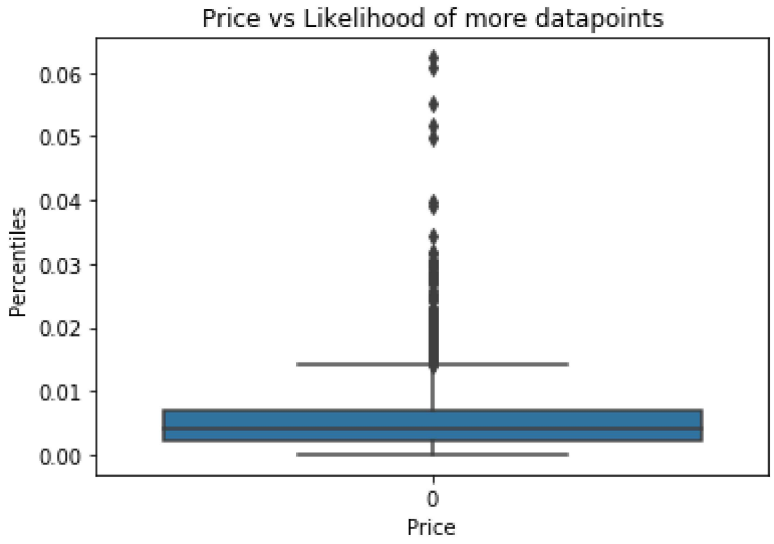
In [45]:

```
#Creating wordcloud from false positive data points of feature essay
X_test_fp_set1 = X_test['essay'][fp_indices]

create_wordcloud_using_fp_essay(X_test_fp_set1)
```



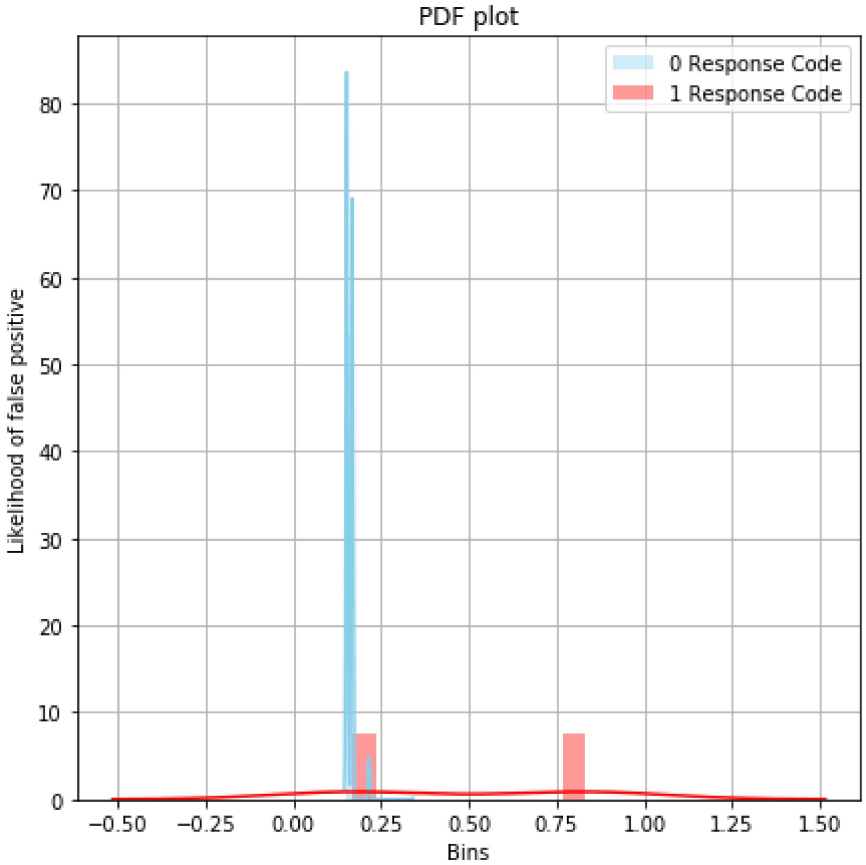
In [46]:

```
#Extracting price feature and plotting Box plot
plot_Box_plot(X_test_price[fp_indices])
```



In [47]:

```
#Extracting teacher_number_of_previously_posted_projects feature and plotting PDF
X_test_tchr_prfx_0_1_fp = np.array(X_test_tchr_prfx_0_1)[fp_indices]

plot_PDF(X_test_tchr_prfx_0_1_fp[:,0], X_test_tchr_prfx_0_1_fp[:1])
```



Train model with Train set 2

In [48]:

```
clf = xgb.XGBClassifier()

#Finding best alpha using GridSearchCV method
clf_set2 = GridSearchCV(estimator = clf, param_grid= param, cv=n_folds, scoring='roc_au
c')
clf_set2.fit(X_tr_set2, y_train)

clf_set2.best_params_
```

Out[48]:

```
{'max_depth': 4, 'n_estimators': 100}
```

Getting Train and CV Scores for set 2

In [49]:

```
lst_train_scores_set2, lst_cv_scores_set2 = getTrain_and_Cv_scores(clf_set2)
```

Printing Train and CV Scores for set 2

In [50]:

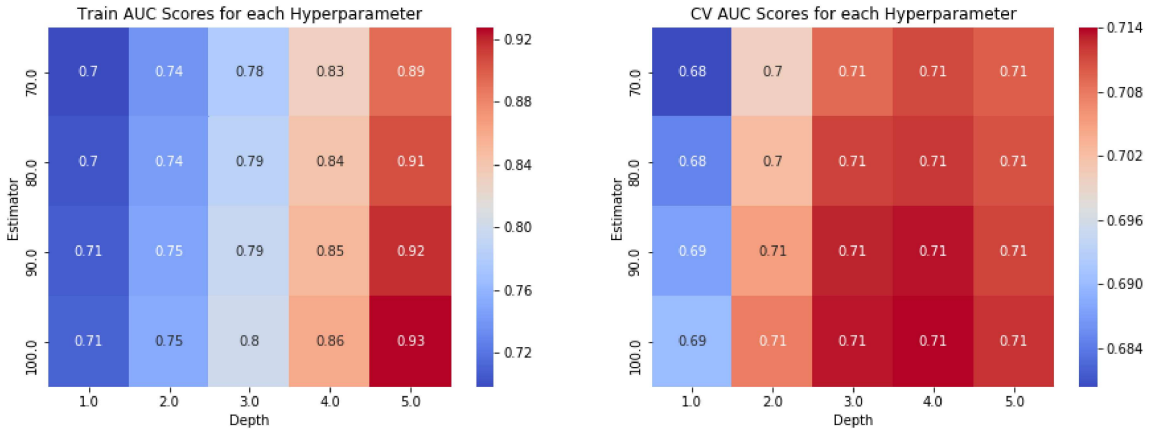
```
print_train_cv_score(lst_train_scores_set2, lst_cv_scores_set2)
```

```
---------Train scores----------
Depth :1 Estimator :70 Auc score:0.6985444324760515
Depth :1 Estimator :80 Auc score:0.7035152872249579
Depth :1 Estimator :90 Auc score:0.7072432444583533
Depth :1 Estimator :100 Auc score:0.7109560983388064
Depth :2 Estimator :70 Auc score:0.7382855217001462
Depth :2 Estimator :80 Auc score:0.7434255212665871
Depth :2 Estimator :90 Auc score:0.7482022245383756
Depth :2 Estimator :100 Auc score:0.752628699398204
Depth :3 Estimator :70 Auc score:0.7796375291085565
Depth :3 Estimator :80 Auc score:0.7877647802185681
Depth :3 Estimator :90 Auc score:0.7948317922991777
Depth :3 Estimator :100 Auc score:0.8011574006068245
Depth :4 Estimator :70 Auc score:0.8314220047988107
Depth :4 Estimator :80 Auc score:0.8422473936999889
Depth :4 Estimator :90 Auc score:0.8520789434922437
Depth :4 Estimator :100 Auc score:0.8606019909684752
Depth :5 Estimator :70 Auc score:0.8930941028615811
Depth :5 Estimator :80 Auc score:0.9051062114466601
Depth :5 Estimator :90 Auc score:0.9168457313271694
Depth :5 Estimator :100 Auc score:0.9271501772244406

---------CV scores----------
Depth :1 Estimator :70 Auc score:0.6803823159109509
Depth :1 Estimator :80 Auc score:0.6847347731276034
Depth :1 Estimator :90 Auc score:0.6874534777200259
Depth :1 Estimator :100 Auc score:0.6902366739587515
Depth :2 Estimator :70 Auc score:0.699947943677166
Depth :2 Estimator :80 Auc score:0.7022139109238704
Depth :2 Estimator :90 Auc score:0.7052001297755816
Depth :2 Estimator :100 Auc score:0.7077657807552441
Depth :3 Estimator :70 Auc score:0.7091407629513536
Depth :3 Estimator :80 Auc score:0.7116081950491848
Depth :3 Estimator :90 Auc score:0.7130112139414283
Depth :3 Estimator :100 Auc score:0.7134805243415716
Depth :4 Estimator :70 Auc score:0.711198583130742
Depth :4 Estimator :80 Auc score:0.7121367014822096
Depth :4 Estimator :90 Auc score:0.7135099098197565
Depth :4 Estimator :100 Auc score:0.7140201936563253
Depth :5 Estimator :70 Auc score:0.7099252737786766
Depth :5 Estimator :80 Auc score:0.7106719558186639
Depth :5 Estimator :90 Auc score:0.7114062343890432
Depth :5 Estimator :100 Auc score:0.712412909123095
```

Plotting 3D graph for Train and CV Scores for set 2

In [51]:

```
plot_HeatMap(lst_train_scores_set2, lst_cv_scores_set2)
```



Training model with best parameter set 2

In [52]:

```
GBDT_clf_set2 = xgb.XGBClassifier(max_depth= clf_set2.best_params_['max_depth'],
                                                  n_estimators= clf_set2.best_params_['n_
estimators'])

GBDT_clf_set2.fit(X_tr_set2, y_train)
```

Out[52]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=4, min_child_weight=1, missing=None,
       n_estimators=100, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
```

Finding AUC curve on train and test data for set 2

In [53]:

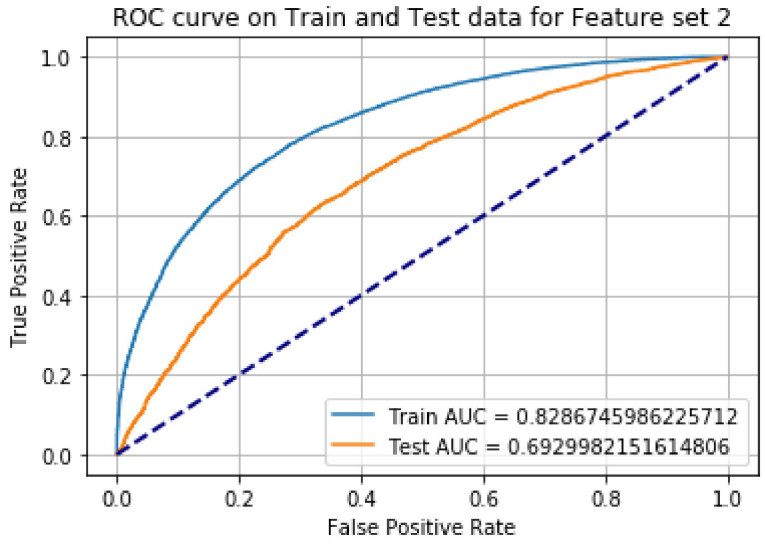
```
y_train_proba_set2 = GBDT_clf_set2.predict_proba(X_tr_set2)[:,1]
y_test_proba_set2 = GBDT_clf_set2.predict_proba(X_te_set2)[:,1]


#Finding AUC on train and test data
train_auc_set2 = roc_auc_score(y_train, y_train_proba_set2)
print('Train Auc for set 2')
print(train_auc_set2)

test_auc_set2 = roc_auc_score(y_test, y_test_proba_set2)
print('\n Test Auc for set 2')
print(test_auc_set2)
```

```
Train Auc for set 2
0.8286745986225712

 Test Auc for set 2
0.6929982151614806
```

Plotting ROC curve of train and test data for set 2

In [54]:

```
#Finding FPR and TPR both on train and test
train_fpr_set2, train_tpr_set2, train_threshold_set2 = roc_curve(y_train, y_train_proba
_set2)
test_fpr_set2, test_tpr_set2, test_threshold_set2 = roc_curve(y_test, y_test_proba_set2
)

#Plotting AUC curve
plot_AUC(train_fpr_set2, train_tpr_set2, test_fpr_set2, test_tpr_set2, train_auc_set2,
test_auc_set2,
         'ROC curve on Train and Test data for Feature set 2')
```



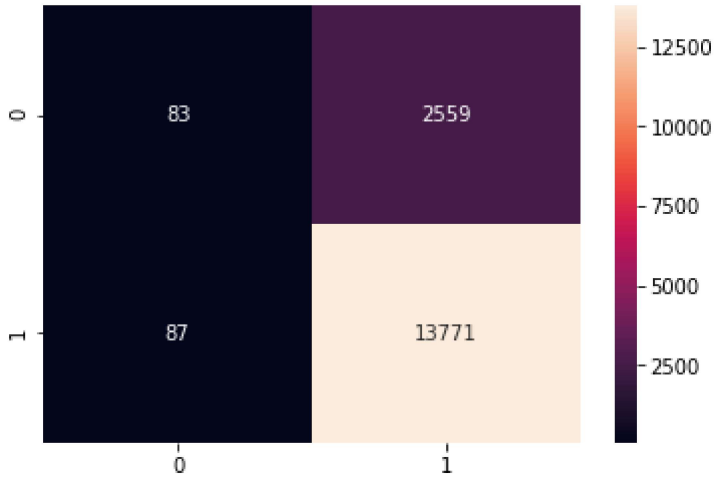
Plotting confusion matrix for set 2

In [55]:

```
#Predicting y_test
y_test_pred_set2 = GBDT_clf_set2.predict(X_te_set2)

confusion_matrix_set2 = confusion_matrix(y_test, y_test_pred_set2)

#Seaborn Heatmap representaion of Train confusion matrix
sns.heatmap(confusion_matrix_set2, annot=True, fmt="d")
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f722d16ba8>
```



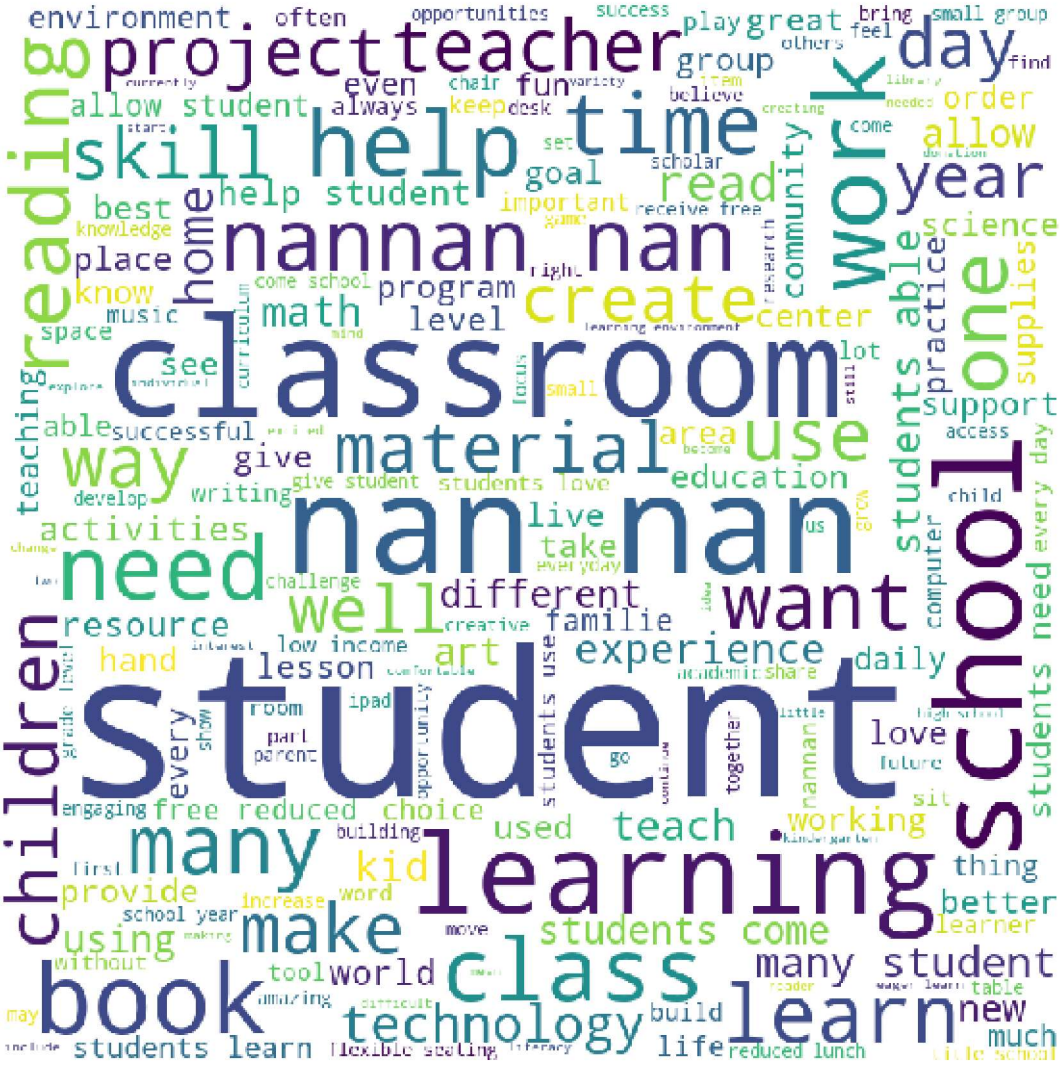
In [56]:

```
#Extracting False positive datapoints

fp_indices = []
for i in range(len(y_test)):
    if (np.array(y_test)[i] == 0) & (y_test_pred_set2[i] == 1):
        fp_indices.append(i)
```

Exploratory Data analysis for set 2

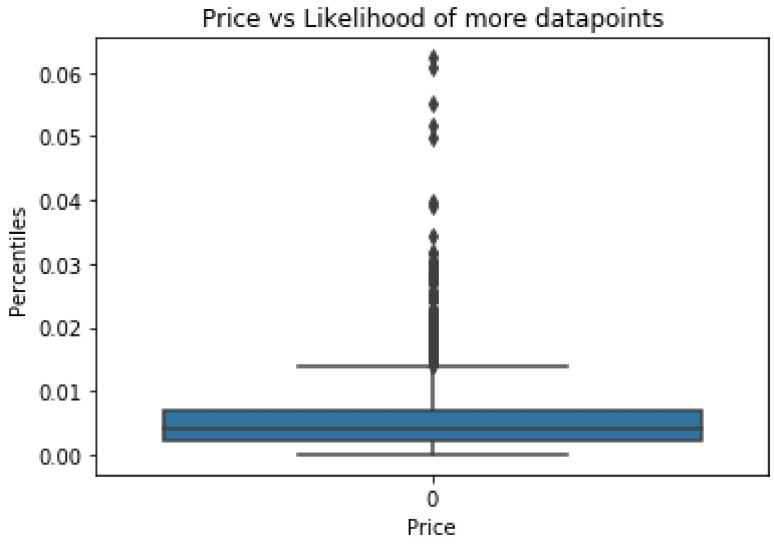
In [57]:

```
#Creating wordcloud from false positive data points of feature essay
X_test_fp_set2 = X_test['essay'][fp_indices]

create_wordcloud_using_fp_essay(X_test_fp_set2)
```



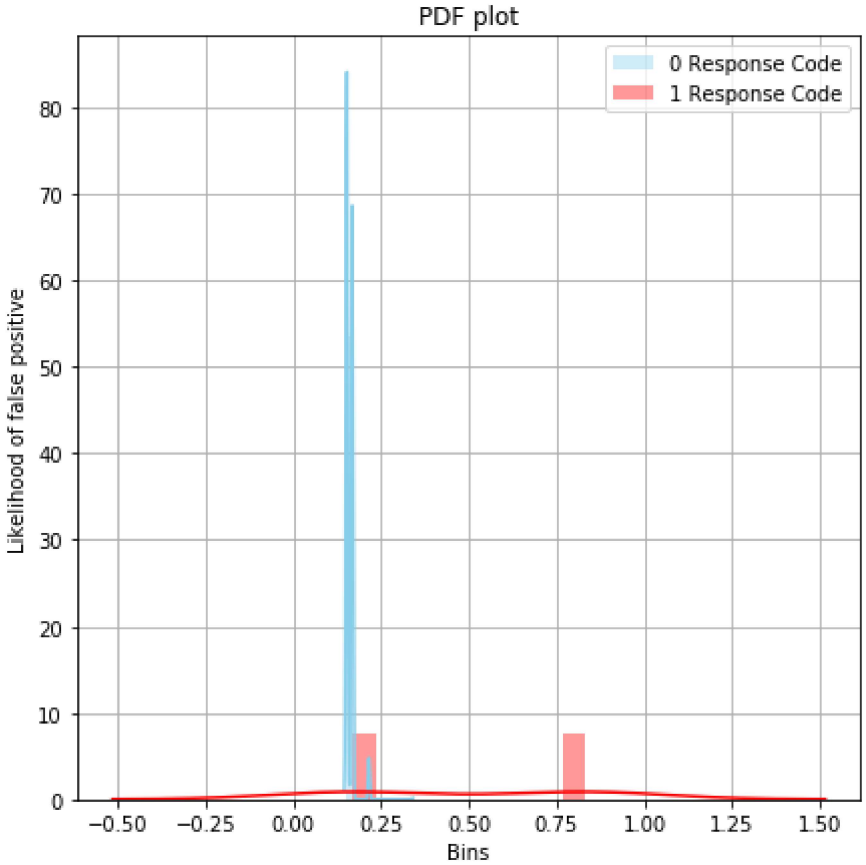
In [58]:

```
#Extracting price feature and plotting Box plot
plot_Box_plot(X_test_price[fp_indices])
```



In [59]:

```
#Extracting teacher_number_of_previously_posted_projects feature and plotting PDF
X_test_tchr_prfx_0_1_fp = np.array(X_test_tchr_prfx_0_1)[fp_indices]

plot_PDF(X_test_tchr_prfx_0_1_fp[:,0], X_test_tchr_prfx_0_1_fp[:1])
```



Summary

In [62]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper parameter", "AUC"]


x.add_row(["TFIDF", 'GBDT', clf_set1.best_params_, test_auc_set2])
x.add_row(["W2V", 'GBDT', clf_set2.best_params_, test_auc_set1])
print(x)
```

```
+------------+-------+--------------------------------------+------------
--------+
| Vectorizer | Model |            Hyper parameter           |        AUC
|
+------------+-------+--------------------------------------+------------
--------+
|   TFIDF    |  GBDT | {'max_depth': 5, 'n_estimators': 100} | 0.692998215
1614806 |
|    W2V     |  GBDT | {'max_depth': 4, 'n_estimators': 100} | 0.701995496
8798374 |
+------------+-------+--------------------------------------+------------
--------+
```