Assignment : DT

Please check twice video before attempting this assignment

```
# Video Machine Learning Basics
```

```
In [1000]: def modelpipeline_grooming_binaries(model):
               # Pipeline for data grooming
               ohe = OneHotEncoder(sparse=False)
               numeric_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='mean')),
                                                     ('scaler', StandardScaler())])
                                                     ('ohe', ohe)])
               categorical_transformer = Pipeline(steps=[
                                                     ('imputer', SimpleImputer(strategy='constant', fill_value='Missing')),
                                                     ('ohe', ohe)])
               preprocessor = ColumnTransformer(
                                                     transformers=[('num', numeric_transformer, selector(dtype_exclude='object')),
                                                     ('cat', categorical_transformer, selector(dtype_include='object'))
                                                     ])
               # Append classifier to preprocessing pipeline.
               # Now we have a full prediction pipeline.
               clf = Pipeline(steps=[('preprocessor', preprocessor),
                                     ('classifier', model)])
               return clf

In [1001]: # Pipeline fitting, hyperparameter search
           ohe = OneHotEncoder(sparse=False)
           numeric_imputer = SimpleImputer(strategy='mean')
           numeric_scaler = StandardScaler()
           categorical_imputer = SimpleImputer(strategy='constant', fill_value='Missing')
           categorical_ohe = OneHotEncoder(sparse=False)
           numeric_transformer = Pipeline(steps=[('imputer', numeric_imputer),
                                                 ('scaler', numeric_scaler)])
                                                 ('ohe', ohe)])
           categorical_transformer = Pipeline(steps=[
                                                 ('imputer', categorical_imputer),
                                                 ('ohe', categorical_ohe)])
           preprocessor = ColumnTransformer(
                                                 transformers=[('num', numeric_transformer, selector(dtype_exclude='object')),
                                                 ('cat', categorical_transformer, selector(dtype_include='object'))
                                                 ])
           # Append classifier to preprocessing pipeline.
           # Now we have a full prediction pipeline.
           clf = Pipeline(steps=[('preprocessor', preprocessor),
                                 ('classifier', model)])
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```

Testing the performance of the model on test data, plotting ROC Curves

```
In [1002]: def test_pipeline(model):
               # Function to fit the pipeline, print the confusion matrix, classification report, and ROC curve
               # for the predicted outputs
               y_pred_proba = clf.fit(X_train, y_train).predict_proba(X_test)[:,1]
               [fpr, tpr, thr] = roc_curve(y_test, y_pred_proba)
               print('Train/Test split results:')
               print(clf.fit(X_train, y_train).score(X_test, y_test))
               y_pred = clf.predict(X_test)
               print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
               print("\n classification report\n", classification_report(y_test, y_pred))
               print("\n AUC score: %.3f" % auc(fpr, tpr))
               print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
               print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
               print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
               print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
               print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
               print("\n Top 5 Features: ")
               pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
               print("\n Bottom 5 Features: ")
               pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
               fig, ax = plt.subplots(figsize=(10, 10))
               plot_confusion_matrix(clf, X_test, y_test, ax=ax)
               plt.show()
               plt.savefig('')

In [1003]: # Function to plot the ROC curve
           def plot_roc(model):
               [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
               plt.plot(fpr, tpr)
               plt.plot([0, 1], [0, 1], 'k--')
               plt.rcParams['font.size'] = 12
               plt.title('ROC curve for Text and Text Score')
               plt.xlabel('False Positive Rate')
               plt.ylabel('True Positive Rate')
```

Plotting the Confusion Matrix for Test Data Point

```
In [1004]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1005]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Testing the performance of the model on test data, plotting ROC Curves

```
In [1006]: def test_pipeline(model):
               # Function to fit the pipeline, print the confusion matrix, classification report, and ROC curve
               # for the predicted outputs
               y_pred_proba = clf.fit(X_train, y_train).predict_proba(X_test)[:,1]
               [fpr, tpr, thr] = roc_curve(y_test, y_pred_proba)
               print('Train/Test split results:')
               print(clf.fit(X_train, y_train).score(X_test, y_test))
               y_pred = clf.predict(X_test)
               print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
               print("\n classification report\n", classification_report(y_test, y_pred))
               print("\n AUC score: %.3f" % auc(fpr, tpr))
               print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
               print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
               print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
               print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
               print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
               print("\n Top 5 Features: ")
               pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
               print("\n Bottom 5 Features: ")
               pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
               fig, ax = plt.subplots(figsize=(10, 10))
               plot_confusion_matrix(clf, X_test, y_test, ax=ax)
               plt.show()
               plt.savefig('')

In [1007]: # Function to plot the ROC curve
           def plot_roc(model):
               [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
               plt.plot(fpr, tpr)
               plt.plot([0, 1], [0, 1], 'k--')
               plt.rcParams['font.size'] = 12
               plt.title('ROC curve for Text and Text Score')
               plt.xlabel('False Positive Rate')
               plt.ylabel('True Positive Rate')
```


Plotting the Confusion Matrix for Test Data Point

```
In [1008]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1009]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Testing the performance of the model on test data, plotting ROC Curves

```
In [1010]: def test_pipeline(model):
               # Function to fit the pipeline, print the confusion matrix, classification report, and ROC curve
               # for the predicted outputs
               y_pred_proba = clf.fit(X_train, y_train).predict_proba(X_test)[:,1]
               [fpr, tpr, thr] = roc_curve(y_test, y_pred_proba)
               print('Train/Test split results:')
               print(clf.fit(X_train, y_train).score(X_test, y_test))
               y_pred = clf.predict(X_test)
               print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
               print("\n classification report\n", classification_report(y_test, y_pred))
               print("\n AUC score: %.3f" % auc(fpr, tpr))
               print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
               print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
               print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
               print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
               print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
               print("\n Top 5 Features: ")
               pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
               print("\n Bottom 5 Features: ")
               pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
               fig, ax = plt.subplots(figsize=(10, 10))
               plot_confusion_matrix(clf, X_test, y_test, ax=ax)
               plt.show()
               plt.savefig('')

In [1011]: # Function to plot the ROC curve
           def plot_roc(model):
               [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
               plt.plot(fpr, tpr)
               plt.plot([0, 1], [0, 1], 'k--')
               plt.rcParams['font.size'] = 12
               plt.title('ROC curve for Text and Text Score')
               plt.xlabel('False Positive Rate')
               plt.ylabel('True Positive Rate')
```


Plotting the Confusion Matrix for Test Data Point

```
In [1012]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1013]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Plotting Confusion Matrix for Test Data Point

```
In [1014]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1015]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Plotting Confusion Matrix for Test Data Point

```
In [1016]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1017]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Plotting Confusion Matrix for Test Data Point

```
In [1018]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1019]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Plotting Confusion Matrix for Test Data Point

```
In [1020]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1021]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Plotting Confusion Matrix for Test Data Point

```
In [1022]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1023]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Plotting Confusion Matrix for Test Data Point

```
In [1024]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1025]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Plotting Confusion Matrix for Test Data Point

```
In [1026]: def find_best_threshold(threshold, fpr, tpr):
               # for the given threshold, fpr, tpr
               t = threshold[np.argmax(tpr*(1-fpr))]
               print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
               return(t)

           def predict_with_best_t(proba, threshold):
               a = []
               for i in proba:
                   if i >= threshold:
                       a.append(1)
                   else:
                       a.append(0)
               return(a)

In [1027]: [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Box Plot with the 'price' of the false positive data points

```
In [1028]: ax = sns.boxplot(
               data=df,
               x='is_exploded_text_classifier',
               y='price',
               hue='is_exploded_text_classifier',
               palette=None,
               order=None,
               hue_order=None,
               orient=None,
               color=None,
               saturation=0.75,
               dodge=True,
               fliersize=5,
               linewidth=None,
               whis=1.5
               ax=None,
               **kwargs
               )
           ax.set(title='Box Plot of Price for False positive Data points')
           plt.savefig('')
```


The PDF with the 'teacher_number_of_previously_posted_projects' of the false positive data points

```
In [1029]: teacher_number_of_previously_posted_projects_1 = df[df['is_exploded_text_classifier'] == 1]['teacher_number_of_previously_p
           sns.distplot(teacher_number_of_previously_posted_projects_1, bins=None, hist=True, kde=True, rug=False, vertical=False, norm
           plt.rcParams['font.size'] = 12
           plt.title('PDF for teacher_number_of_previously_posted_projects for False positive Data points')
           plt.xlabel('teacher_number_of_previously_posted_projects')
           plt.ylabel('Density')
           plt.savefig('')
```


Hyper Parameter Tuning using GridSearchCV on Set-2 (categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)

```
In [1030]: from sklearn.model_selection import GridSearchCV
           from sklearn.ensemble import GradientBoostingClassifier
           from sklearn.tree import DecisionTreeClassifier
           from sklearn.metrics import roc_curve, auc
           import numpy as np
           import warnings
           warnings.filterwarnings("ignore")
           [fpr, tpr, thr] = roc_curve(y_test, clf.fit(X_train, y_train).predict_proba(X_test)[:,1])
           print("Train/Test split results:")
           print(clf.fit(X_train, y_train).score(X_test, y_test))
           y_pred = clf.predict(X_test)
           print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
           print("\n classification report\n", classification_report(y_test, y_pred))
           print("\n AUC score: %.3f" % auc(fpr, tpr))
           print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
           print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
           print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
           print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y_test, y_pred))
           print("\n matthews_corrcoef: %.3f" % matthews_corrcoef(y_test, y_pred))
           print("\n Top 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[:5].
           print("\n Bottom 5 Features: ")
           pd.DataFrame(zip(X_train.columns, clf.fit(X_train, y_train).feature_importances_).sort_values(by=1, ascending=False)[-5:].
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


3-D plot representing the performance for train and cross validation data for each value of hyper parameters

```
In [1031]: modelpipeline_grooming_binaries(model):
               # Pipeline for data grooming
               ohe = OneHotEncoder(sparse=False)
               numeric_transformer = Pipeline(steps=[('imputer', SimpleImputer(strategy='mean')),
                                                     ('scaler', StandardScaler())])
                                                     ('ohe', ohe)])
               categorical_transformer = Pipeline(steps=[
                                                     ('imputer', SimpleImputer(strategy='constant', fill_value='Missing')),
                                                     ('ohe', ohe)])
               preprocessor = ColumnTransformer(
                                                     transformers=[('num', numeric_transformer, selector(dtype_exclude='object')),
                                                     ('cat', categorical_transformer, selector(dtype_include='object'))
                                                     ])
               # Append classifier to preprocessing pipeline.
               # Now we have a full prediction pipeline.
               clf = Pipeline(steps=[('preprocessor', preprocessor),
                                     ('classifier', model)])
               return clf

In [1032]: # Pipeline fitting, hyperparameter search
           ohe = OneHotEncoder(sparse=False)
           numeric_imputer = SimpleImputer(strategy='mean')
           numeric_scaler = StandardScaler()
           categorical_imputer = SimpleImputer(strategy='constant', fill_value='Missing')
           categorical_ohe = OneHotEncoder(sparse=False)
           numeric_transformer = Pipeline(steps=[('imputer', numeric_imputer),
                                                 ('scaler', numeric_scaler)])
                                                 ('ohe', ohe)])
           categorical_transformer = Pipeline(steps=[
                                                 ('imputer', categorical_imputer),
                                                 ('ohe', categorical_ohe)])
           preprocessor = ColumnTransformer(
                                                 transformers=[('num', numeric_transformer, selector(dtype_exclude='object')),
                                                 ('cat', categorical_transformer, selector(dtype_include='object'))
                                                 ])
           # Append classifier to preprocessing pipeline.
           # Now we have a full prediction pipeline.
           clf = Pipeline(steps=[('preprocessor', preprocessor),
                                 ('classifier', model)])
           fig, ax = plt.subplots(figsize=(10, 10))
           plot_confusion_matrix(clf, X_test, y_test, ax=ax)
           plt.show()
           plt.savefig('')
```


Testing the performance of the model on test data, plotting ROC Curves

```
In [1033]: def test_pipeline(model):
               # Function to fit the pipeline, print the confusion matrix, classification report, and ROC curve
               # for the predicted outputs
               y_pred_proba = clf.fit(X_train, y_train).predict_proba(X_test)[:,1]
               [fpr, tpr, thr] = roc_curve(y_test, y_pred_proba)
               print('Train/Test split results:')
               print(clf.fit(X_train, y_train).score(X_test, y_test))
               y_pred = clf.predict(X_test)
               print("\n confusion matrix\n", confusion_matrix(y_test, y_pred))
               print("\n classification report\n", classification_report(y_test, y_pred))
               print("\n AUC score: %.3f" % auc(fpr, tpr))
               print("\n accuracy_score: %.3f" % accuracy_score(y_test, y_pred))
               print("\n average_precision_score: %.3f" % average_precision_score(y_test, y_pred))
               print("\n f1_score: %.3f" % f1_score(y_test, y_pred))
               print("\n Cohen Kappa Score: %.3f" % cohen_kappa_score(y
```

In [109]:
```
def visualize_proba_threshold(clf,
                              X_train,
                              y_train,
                              X_test,
                              y_test,
                              threshold=0.5,
                              pos_label=1):
    '''
    Plot the model's predicted probabilities for
    the positive class (pos_label) on the test set,
    with a vertical line at the given threshold.
    Also prints accuracy, precision, recall, and
    confusion matrix at that threshold.
    '''
    clf.fit(X_train, y_train)

    y_test_proba = clf.predict_proba(X_test)[:,1]

    y_test_pred = (y_test_proba >= threshold).astype(int)

    print("Threshold:", threshold)
    print("Accuracy:", metrics.accuracy_score(y_test, y_test_pred))
    print("Precision:", metrics.precision_score(y_test, y_test_pred, pos_label=pos_label))
    print("Recall:", metrics.recall_score(y_test, y_test_pred, pos_label=pos_label))
    print("Confusion matrix\n", metrics.confusion_matrix(y_test, y_test_pred))

    fpr, tpr, thresholds = metrics.roc_curve(y_test, y_test_proba, pos_label=pos_label)

    plt.plot(fpr, tpr, '-')
    plt.plot([0, 1], [0, 1], '--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.show()
```

In [110]:
```
pipe1 =
pipe_tfidf,
pipe_cvec,
pipe_hash,
pipe_cvec_tfidf,
pipe_hash_tfidf,
]

for pipeline in pipeline_list:
    print('---------------------------------')
    print('Pipeline:', pipeline.steps[-1][0], sep='\n')
    pipeline.fit(X_train, y_train)
    y_test = pipeline.predict(X_test)
    y_test = y_test.astype(int)
    print('Accuracy:', metrics.accuracy_score(y_test, y_test))
    print('Precision:', metrics.precision_score(y_test, y_test))
    print('Recall:', metrics.recall_score(y_test, y_test))
    print('Confusion matrix\n', metrics.confusion_matrix(y_test, y_test))
    print('---------------------------------')
```

```
---------------------------------
Pipeline:
tfidf
Accuracy: 0.8
Precision: 0.8
Recall: 1.0
Confusion matrix
 [[ 0 20]
 [ 0 80]]
---------------------------------
---------------------------------
Pipeline:
cvec
Accuracy: 0.8
Precision: 0.8
Recall: 1.0
Confusion matrix
 [[ 0 20]
 [ 0 80]]
---------------------------------
---------------------------------
Pipeline:
hash
Accuracy: 0.8
Precision: 0.8
Recall: 1.0
Confusion matrix
 [[ 0 20]
 [ 0 80]]
---------------------------------
---------------------------------
Pipeline:
cvec_tfidf
Accuracy: 0.8
Precision: 0.8
Recall: 1.0
Confusion matrix
 [[ 0 20]
 [ 0 80]]
---------------------------------
---------------------------------
Pipeline:
hash_tfidf
Accuracy: 0.8
Precision: 0.8
Recall: 1.0
Confusion matrix
 [[ 0 20]
 [ 0 80]]
---------------------------------
```

In [111]:
```
pipe_tfidf.fit(X_train, y_train)
y_test = pipe_tfidf.predict(X_test)
y_test = y_test.astype(int)
print('Accuracy:', metrics.accuracy_score(y_test, y_test))
print('Precision:', metrics.precision_score(y_test, y_test))
print('Recall:', metrics.recall_score(y_test, y_test))
print('Confusion matrix\n', metrics.confusion_matrix(y_test, y_test))
```

```
Accuracy: 0.8
Precision: 0.8
Recall: 1.0
Confusion matrix
 [[ 0 20]
 [ 0 80]]
```

In [112]:
```
visualize_proba_threshold(pipe_tfidf,
                          X_train,
                          y_train,
                          X_test,
                          y_test,
                          threshold=0.5,
                          pos_label=1)
```

```
Threshold: 0.5
Accuracy: 0.8
Precision: 0.8
Recall: 1.0
Confusion matrix
 [[ 0 20]
 [ 0 80]]
```


Testing the performance of the model on test data, plotting ROC Curves

In [113]:
```
def plot_proba_predictions(clf, X_train,
                           y_train, X_test,
                           y_test, pos_label=1):
    '''
    Plot the model's predicted probabilities for the test
    positive class (pos_label) on the test set.
    '''
    clf.fit(X_train, y_train)

    y_test_proba = clf.predict_proba(X_test)[:,1]

    plt.hist(y_test_proba[y_test==pos_label],
             bins=20,
             alpha=0.5,
             color='red',
             label='Positive')
    plt.hist(y_test_proba[y_test!=pos_label],
             bins=20,
             alpha=0.5,
             color='blue',
             label='Negative')
    plt.xlabel('Predicted probability')
    plt.ylabel('Count')
    plt.title('Test Data Predicted Probabilities')
    plt.legend(loc='upper center')
    plt.show()
```

In [114]:
```
clf_lr = LogisticRegression(C=1e9, max_iter=1e4,
                            random_state=42)

y_test_proba = clf_lr.fit(X_train, y_train).
                 predict_proba(X_test)[:,1]

fpr, tpr, thresholds = metrics.roc_curve(y_test,
                         y_test_proba,
                         pos_label=1)

plt.plot(fpr, tpr, '-')
plt.plot([0, 1], [0, 1], '--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```

Plotting the Confusion Matrix for Test Data Point

In [115]:
```
def plot_proba_threshold(clf, X_train,
                         y_train, X_test,
                         y_test, pos_label=1):
    '''
    Plot the model's predicted probabilities for
    the positive class (pos_label) on the test set,
    with a vertical line at the given threshold.
    '''
    clf.fit(X_train, y_train)

    y_test_proba = clf.predict_proba(X_test)[:,1]

    plt.hist(y_test_proba[y_test==pos_label],
             bins=20,
             alpha=0.5,
             color='red',
             label='Positive')
    plt.hist(y_test_proba[y_test!=pos_label],
             bins=20,
             alpha=0.5,
             color='blue',
             label='Negative')
    plt.xlabel('Predicted probability')
    plt.ylabel('Count')
    plt.title('Test Data Predicted Probabilities')
    plt.legend(loc='upper center')
    plt.show()
```

In [116]:
```
plot_proba_threshold(clf_lr, X_train,
                     y_train, X_test,
                     y_test, pos_label=1)
```


Collecting the False Positive Row Indexes

In [117]:
```
fp_rows_idx = np.where(y_test == 0)[0][y_test[y_test==0].astype(int).ravel() == y_test_pred[y_test==0]]
fp_rows_idx[:5] # show first five false positive rows
fp_rows_idx = np.where(y_test == 0)[0][y_test[y_test==0].astype(int).ravel() != y_test_pred[y_test==0]]
fp_rows_idx[:5] # show first five false positive rows
```

```
array([ 2, 12, 14, 18, 20])
array([ 0, 1, 3, 4, 5])
```

WordCloud with the words of 'essay' text of these false positive data points

In [118]:
```
text_index = 0
for idx in fp_rows_idx:
    text_index += 1
    print('Text', text_index, sep=' ')
    print(X_test[idx])
    print('\n')
%matplotlib inline
from wordcloud import WordCloud
wc = WordCloud(stopwords=stopwords,
               background_color='white',
               max_words=200,
               collocations=False)
plt.imshow(wc.generate(' '.join(X_test[fp_rows_idx])))
plt.axis('off')
plt.title('Word Cloud of Words from False positive data points')
plt.show()
```

```
Text 1
I teach 3rd grade in Title 1 schools. Most of my students are English Language Learners. Many of my students live in poverty. Many of my students live in single parent homes. Many of my students live with their grandparents. Many of my students live in foster homes. Many of my students live in shelters. Many of my students have parents who are incarcerated. Many of my students have parents who are addicted to drugs and alcohol. Many of my students have parents who are in the military. Many of my students have parents who are deployed. Many of my students have parents who are disabled. Many of my students have parents who are unemployed. Many of my students have parents who are underemployed. Many of my students have parents who are homeless. Many of my students have parents who are immigrants. Many of my students have parents who are refugees. Many of my students have parents who are victims of domestic violence. Many of my students have parents who are victims of human trafficking. Many of my students have parents who are victims of gang violence. Many of my students have parents who are victims of sex trafficking. Many of my students have parents who are victims of drive by shootings. Many of my students have parents who are victims of physical abuse. Many of my students have parents who are victims of sexual abuse. Many of my students have parents who are victims of mental abuse. Many of my students have parents who are victims of emotional abuse. Many of my students have parents who are victims of verbal abuse. Many of my students have parents who are victims of homicide. Many of my students have parents who are victims of suicide. Many of my students have parents who are victims of terrorism. Many of my students have parents who are victims of war. Many of my students have parents who are victims of genocide. Many of my students have parents who are victims of ethnic cleansing. Many of my students have parents who are victims of racial profiling. Many of my students have parents who are victims of discrimination. Many of my students have parents who are victims of bullying. Many of my students have parents who are victims of cyberbullying. Many of my students have parents who are victims of hate crimes. Many of my students have parents who are victims of sex crimes. Many of my students have parents who are victims of child abuse. Many of my students have parents who are victims of elder abuse. Many of my students have parents who are victims of animal abuse. Many of my students have parents who are victims of environmental abuse. Many of my students have parents who are victims of political abuse. Many of my students have parents who are victims of religious abuse. Many of my students have parents who are victims of cultural abuse. Many of my students have parents who are victims of racial abuse. Many of my students have parents who are victims of ethnic abuse. Many of my students have parents who are victims of gender abuse. Many of my students have parents who are victims of sexual orientation abuse. Many of my students have parents who are victims of disability abuse. Many of my students have parents who are victims of age
```