Part 1. Dataset Assessment

The dataset consists of 14640 observations and has 15 features including the target variable, airline_sentiment. Of those features, airline_sentiment_gold, negativereason_gold, and tweet_coord have above 90% of null values, so it is better to neglect those columns when building a prediction model. The tweet_location and user_timezone features have more than 30% of null values. Even with the data that is not null, tweet_location has a lot of invalid data like 'this place called NYC', or 'Somewhere celebrating life,' which cannot be used for prediction models. Similarly, user_timezone has too much variety but very skewed data with most data being in the US & Canada timezone. Those features are also neglected when choosing the predictor variables. Also, nagativereason and negativereason_confidence are also neglected to keep the dataset large enough because they have around 30% of null values.

```
print("Percentage null or na values in df")
((data.isna()).sum() * 100 / data.index.size).round(2)
```

```
Percentage null or na values in df

tweet_id                          0.00
airline_sentiment                 0.00
airline_sentiment_confidence      0.00
negativereason                   37.31
negativereason_confidence        28.13
airline                           0.00
airline_sentiment_gold           99.73
name                              0.00
negativereason_gold              99.78
retweet_count                     0.00
text                              0.00
tweet_coord                      93.04
tweet_created                     0.00
tweet_location                   32.33
user_timezone                    32.92
dtype: float64
```

When using tweet_created variable, it required some feature engineering because it is a object type variable and the predictive models cannot function with object or string type variables. I divided the tweet_created column into date and time columns and removed anything other than the numeric variables. Date column now have a format of [YYYYMMDD] and time column now have a format of [HHMMSS]. The airline column is made into dummy variables and have value of 1 if they are in the category of the airline.

Part 2. Action List of Steps

Data Preparation. Using TextBlob package, subjectivity and polarity scores are evaluated through sentiment analysis process. Polarity shows how negative or positive a text is, and it is represented by continuous numerical value between 1 and -1, 1 being positive and -1 being negative. Subjectivity shows how objective or subjective a text is, and it is represented by continuous numerical value between 0 and 1, 0 being objective and 1 being subjective. Then I split the dataset into 66% train set and 33% test set.

Building Model. I built logistic regression model, decision tree, and random forest without any adjustment and fit models into train set. Then I evaluated accuracy score. To use grid search, I put various range of parameters into GridSearchCV for each model and fit the model into train set. When I got which parameter scores the best, I created another model for each and evaluated the score.

Evaluating Model. To evaluate model performance, I used the score() function, confusion matrix method, and classification_report() function to see the precision score, recall score, and f1-score.

Code:
TextBlob:

```
from textblob import TextBlob
# polarity (positive to negative: 1 to -2)
# subjectivity (objective to subjective: 0 to 1)
```

```
# Create a functino to get the subjectivity
def getSubjectivity(text):
    return TextBlob(text).sentiment.subjectivity

#Create a function to get the polarity
def getPolarity(text):
    return TextBlob(text).sentiment.polarity

#Create two new columns 'Subjectivity' & 'Polarity'
data['subjectivity'] = data['text'].apply(getSubjectivity)
data['polarity'] = data['text'].apply(getPolarity)
data.head()
```

Models with no adjustments:

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
print(lr.score(X_train, y_train))
print(lr.score(X_test, y_test))

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
print(clf.score(X_train, y_train))
print(clf.score(X_test, y_test))

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
print(rf.score(X_train, y_train))
print(rf.score(X_test, y_test))
```

```
0.6212275693311582
0.6384519867549668
0.9989804241435563
0.5767798013245033
0.9989804241435563
0.6316225165562914
```

GridSearch and HyperParameter:

Logistic Regression

```
lr = LogisticRegression(solver='lbfgs', multi_class='multinomial', max_ite
grid={"C":np.logspace(-3,3,20), "penalty":["l1","l2"]}# l1 lasso l2 ridge
lr_cv=GridSearchCV(lr,grid,cv=10, scoring='accuracy', refit=True)
lr_cv.fit(X_train, y_train)

print("tuned hpyerparameters :(best parameters) ",lr_cv.best_params_)
print("accuracy :",lr_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.001, 'penalty': 'l2'}
accuracy : 0.6212276103101791
```

```
lr2 = LogisticRegression(C=0.001, penalty="l2")
lr2.fit(X_train, y_train)
print(lr2.score(X_train, y_train))
print(lr2.score(X_test, y_test))
```

```
0.6212275693311582
0.6384519867549668
```

Decision Tree

```
param_grid = [{
    'max_depth': [2, 4, 8, 16, 32, 64],
    'min_samples_leaf': [2, 4, 8, 16]
}]

clf = DecisionTreeClassifier()

np.random.seed(1)
clf_gridsearch = GridSearchCV(clf, param_grid, cv=10,
                              scoring='accuracy', refit=True)
clf_gridsearch.fit(X_train, y_train)
print(clf_gridsearch.best_score_)
print(clf_gridsearch.best_params_)
```

```
0.6637406644615034
{'max_depth': 8, 'min_samples_leaf': 16}
```

```
clf_model = clf_gridsearch.best_estimator_
print('Training Score:', clf_model.score(X_train, y_train))
print('Testing Score:', clf_model.score(X_test, y_test))
```

```
Training Score: 0.6912724306688418
Testing Score: 0.6668046357615894
```

Random Forest

```
param_grid = [{
    'max_depth':[2, 4, 8, 16, 32, 64, 128, 256], # 128, 256 added
    'min_samples_leaf':[2, 4, 8, 16]
}]

rf = RandomForestClassifier(n_estimators=200)
np.random.seed(1)

rf_gridsearch = GridSearchCV(rf, param_grid, cv=10,
                             scoring='accuracy', refit=True)
rf_gridsearch.fit(X_train, y_train)

print(rf_gridsearch.best_score_)
print(rf_gridsearch.best_params_)
```

```
0.674547005346481
{'max_depth': 64, 'min_samples_leaf': 8}
```

```
rf_model = rf_gridsearch.best_estimator_
print('Training Score:', rf_model.score(X_train, y_train))
print('Testing Score:', rf_model.score(X_test, y_test))
```

```
Training Score: 0.7203303425774877
Testing Score: 0.6738410596026491
```

Feature Importance:

```
# get importance
importance = rf_model.feature_importances_

# Sort the feature importance in descending order
sorted_indices = np.argsort(importance)[::-1]

feat_labels = predictors

for f in range(X_train.shape[1]):
    print("%2d) %-*s %f" % (f + 1, 30,
                            feat_labels[sorted_indices[f]],
                            importance[sorted_indices[f]]))
```

```
 1) polarity                       0.406032
 2) time                           0.202528
 3) subjectivity                   0.172344
 4) date                           0.058280
 5) airline_Delta                  0.048184
 6) airline_US Airways             0.032292
 7) airline_Southwest              0.026363
 8) airline_Virgin America         0.021803
 9) airline_United                 0.016616
10) airline_American               0.015558
```