Flight Delay Prediction

Project 2 - Elvin

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
import matplotlib.pylab as plt
from dmba import classificationSummary, gainsChart
import ppscore as pps
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
```

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

Importing file

In [36]:
```
# Read the data file
delays_df = pd.read_csv('FlightDelays.csv')
delays_dftemp = delays_df
```

In [37]:
```
#Present 5 data records
delays_df.head(5)
```

Out[37]:

	CRS_DEP_TIME	CARRIER	DEP_TIME	DEST	DISTANCE	FL_DATE	FL_NUM	ORIGIN	Weathe
0	1455	OH	1455	JFK	184	37987	5935	BWI	
1	1640	DH	1640	JFK	213	2004-01-01	6155	DCA	
2	1245	DH	1245	LGA	229	2004-01-01	7208	IAD	
3	1715	DH	1709	LGA	229	2004-01-01	7215	IAD	
4	1039	DH	1035	LGA	229	2004-01-01	7792	IAD	

In [38]:
```
nameDict={"Flight Status":"Flight_Status"}
delays_df=delays_df.rename(columns=nameDict)
```

In [39]:
```
delays_df.head(5)
```

Out[39]:

	CRS_DEP_TIME	CARRIER	DEP_TIME	DEST	DISTANCE	FL_DATE	FL_NUM	ORIGIN	Weath
0	1455	OH	1455	JFK	184	37987	5935	BWI	
1	1640	DH	1640	JFK	213	2004-01-01	6155	DCA	
2	1245	DH	1245	LGA	229	2004-01-01	7208	IAD	
3	1715	DH	1709	LGA	229	2004-01-01	7215	IAD	
4	1039	DH	1035	LGA	229	2004-01-01	7792	IAD	

In [40]:
```
delays_df.shape
delays_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2201 entries, 0 to 2200
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   CRS_DEP_TIME   2201 non-null   int64 
 1   CARRIER        2201 non-null   object
 2   DEP_TIME       2201 non-null   int64 
 3   DEST           2201 non-null   object
 4   DISTANCE       2201 non-null   int64 
 5   FL_DATE        2201 non-null   object
 6   FL_NUM         2201 non-null   int64 
 7   ORIGIN         2201 non-null   object
 8   Weather        2201 non-null   int64 
 9   DAY_WEEK       2201 non-null   int64 
 10  DAY_OF_MONTH   2201 non-null   int64 
 11  TAIL_NUM       2201 non-null   object
 12  Flight_Status  2201 non-null   object
dtypes: int64(7), object(6)
memory usage: 223.7+ KB
```

Refrence matrix of the non numeric values which will be converter to numberic values

In [42]:
```
tp=delays_df.CARRIER.unique()
fp = list(range(1, len(tp)+1))
print("Reference for CARRIER")
np.column_stack((tp, fp))
```

```
Reference for CARRIER
```

Out[42]:
```
array([['OH', 1],
       ['DH', 2],
       ['DL', 3],
       ['MQ', 4],
       ['UA', 5],
       ['US', 6],
       ['RU', 7],
       ['CO', 8]], dtype=object)
```

In [43]:
```
tp=delays_df.DEST.unique()
fp = list(range(1, len(tp)+1))
print("Reference for CARRIER")
np.column_stack((tp, fp))
```

```
Reference for CARRIER
```

Out[43]:
```
array([['JFK', 1],
       ['LGA', 2],
       ['EWR', 3]], dtype=object)
```

In [44]:
```
tp=delays_df.ORIGIN.unique()
fp = list(range(1, len(tp)+1))
print("Reference for CARRIER")
np.column_stack((tp, fp))
```

```
Reference for CARRIER
```

Out[44]:
```
array([['BWI', 1],
       ['DCA', 2],
       ['IAD', 3]], dtype=object)
```

In [45]:
```
tp=delays_df.TAIL_NUM.unique()
fp = list(range(1, len(tp)+1))
print("Reference for CARRIER")
np.column_stack((tp, fp))
```

```
Reference for CARRIER
```

Out[45]:
```
array([['N940CA', 1],
       ['N405FJ', 2],
       ['N695BR', 3],
       ...,
       ['N934DL', 547],
       ['N592UA', 548],
       ['N15932', 549]], dtype=object)
```

In [46]:
```
tp=delays_df.Flight_Status.unique()
fp = list(range(1, len(tp)+1))
print("Reference for CARRIER")
np.column_stack((tp, fp))
```

```
Reference for CARRIER
```

Out[46]:
```
array([['ontime', 1],
       ['delayed', 2]], dtype=object)
```

In [47]:
```
tp=delays_df.FL_DATE.unique()
fp = list(range(1, len(tp)+1))
print("Reference for CARRIER")
np.column_stack((tp, fp))
```

```
Reference for CARRIER
```

Out[47]:
```
array([['37987', 1],
       ['2004-01-01', 2],
       ['2004-01-02', 3],
       ['2004-01-03', 4],
       ['2004-01-04', 5],
       ['2004-01-05', 6],
       ['2004-01-06', 7],
       ['2004-01-07', 8],
       ['2004-01-08', 9],
       ['2004-01-09', 10],
       ['2004-01-10', 11],
       ['2004-01-11', 12],
       ['2004-01-12', 13],
       ['2004-01-13', 14],
       ['2004-01-14', 15],
       ['2004-01-15', 16],
       ['2004-01-16', 17],
       ['2004-01-17', 18],
       ['2004-01-18', 19],
       ['2004-01-19', 20],
       ['2004-01-20', 21],
       ['2004-01-21', 22],
       ['2004-01-22', 23],
       ['2004-01-23', 24],
       ['2004-01-24', 25],
       ['2004-01-25', 26],
       ['2004-01-26', 27],
       ['2004-01-27', 28],
       ['2004-01-28', 29],
       ['2004-01-29', 30],
       ['2004-01-30', 31],
       ['2004-01-31', 32]], dtype=object)
```

converting non numeric values into numeric values

In [49]:
```
# Datapreprocessing
l1=LabelEncoder()
#perform label encoding on 'team' column
delays_df['CARRIER'] = l1.fit_transform(delays_df['CARRIER'])
delays_df['DEST'] = l1.fit_transform(delays_df['DEST'])
delays_df['ORIGIN'] = l1.fit_transform(delays_df['ORIGIN'])
delays_df['TAIL_NUM'] = l1.fit_transform(delays_df['TAIL_NUM'])
delays_df['Flight_Status'] = l1.fit_transform(delays_df['Flight_Status'])
delays_df['FL_DATE'] = l1.fit_transform(delays_df['FL_DATE'])
```

In [50]:
```
delays_df
```

Out[50]:

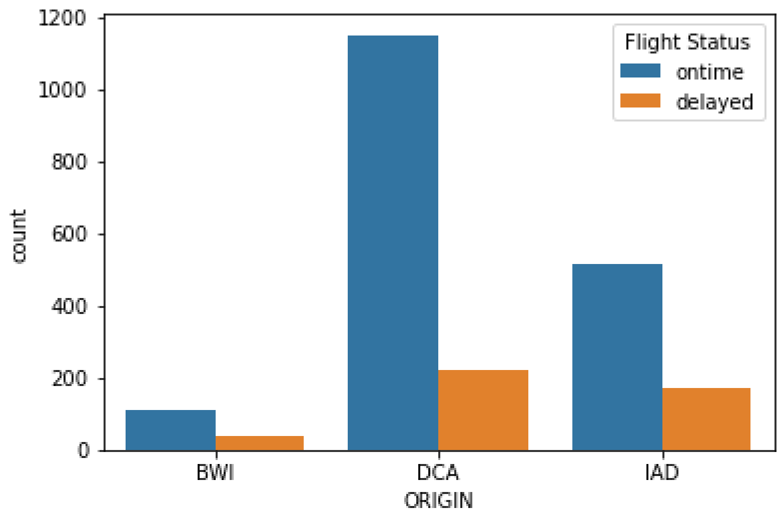
	CRS_DEP_TIME	CARRIER	DEP_TIME	DEST	DISTANCE	FL_DATE	FL_NUM	ORIGIN	We
0	1455	4	1455	1	184	31	5935	0	
1	1640	1	1640	1	213	0	6155	1	
2	1245	1	1245	2	229	0	7208	2	
3	1715	1	1709	2	229	0	7215	2	
4	1039	1	1035	2	229	0	7792	2	
...	...	...	...	...	...	...	...	...	
2196	645	5	644	0	199	30	2761	1	
2197	1700	5	1653	0	213	30	2497	2	
2198	1600	5	1558	0	199	30	2361	1	
2199	1359	5	1403	0	199	30	2216	1	
2200	1730	5	1736	0	199	30	2097	1	

2201 rows × 13 columns

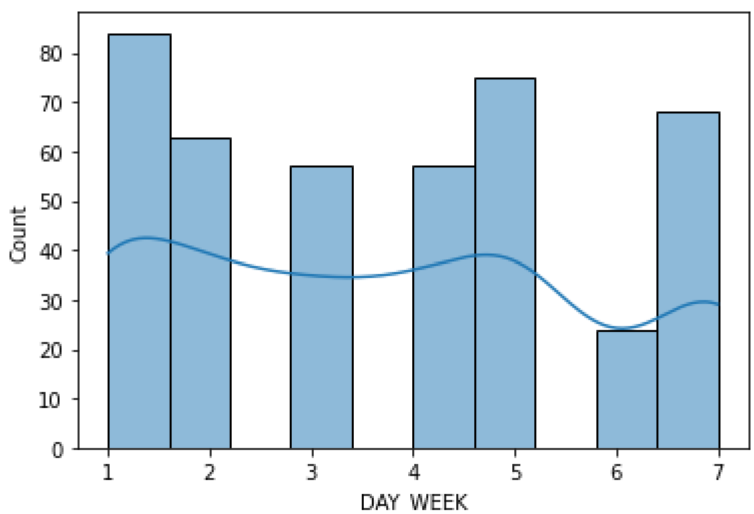
Data Exploration

In [51]:
```
sns.countplot(data=delays_dftemp, x="ORIGIN", hue="Flight Status")
```

Out[51]:
```
<AxesSubplot:xlabel='ORIGIN', ylabel='count'>
```

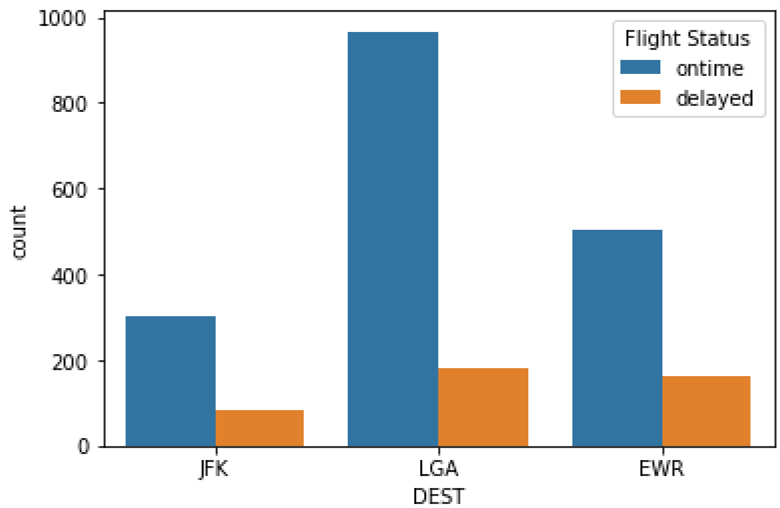


In [56]:
```
delayedflights = delays_dftemp.loc[delays_dftemp['Flight Status'] == "delayed"
histplt = sns.histplot(data=delayedflights, x="DAY_WEEK", kde=True)
plt.show()
```



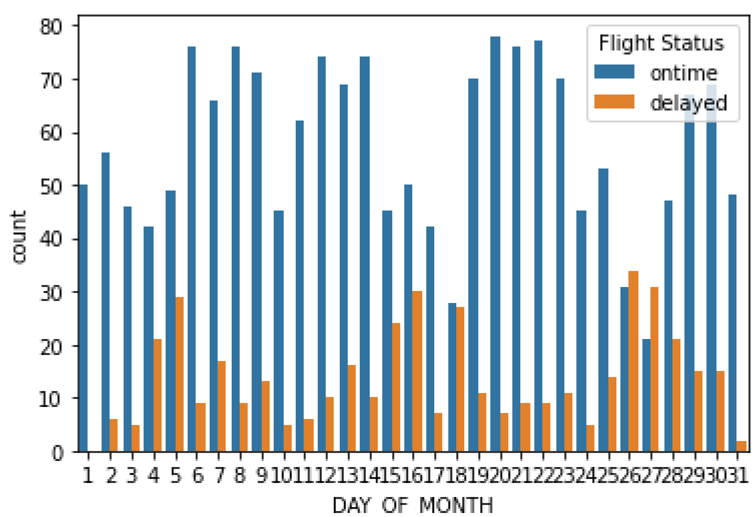
In [105]:
```
sns.countplot(data=delays_dftemp, x="DEST", hue="Flight Status")
```

Out[105]: <AxesSubplot:xlabel='DEST', ylabel='count'>



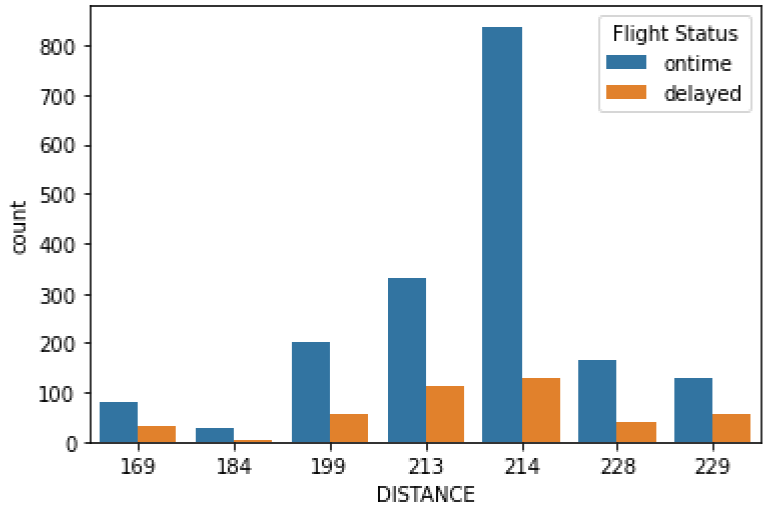
In [106]:
```
sns.countplot(data=delays_dftemp, x="DAY_OF_MONTH", hue="Flight Status")
```

Out[106]: <AxesSubplot:xlabel='DAY_OF_MONTH', ylabel='count'>

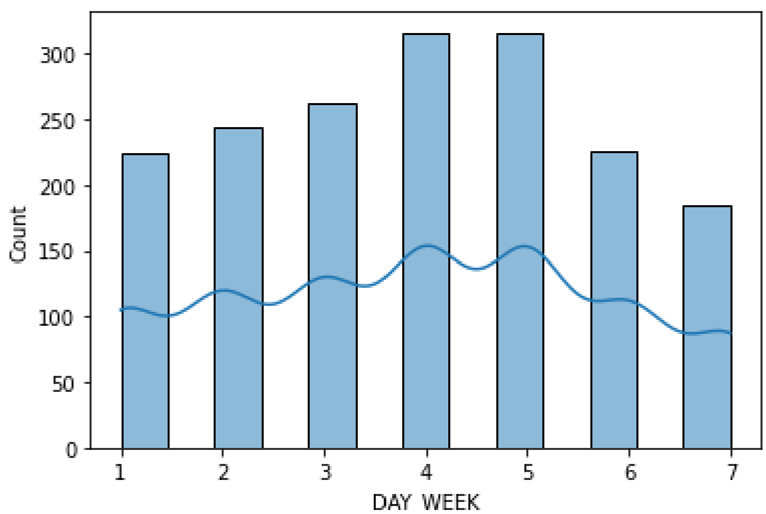


In [104]:
```
sns.countplot(data=delays_dftemp, x="DISTANCE", hue="Flight Status")
```

Out[104]: <AxesSubplot:xlabel='DISTANCE', ylabel='count'>

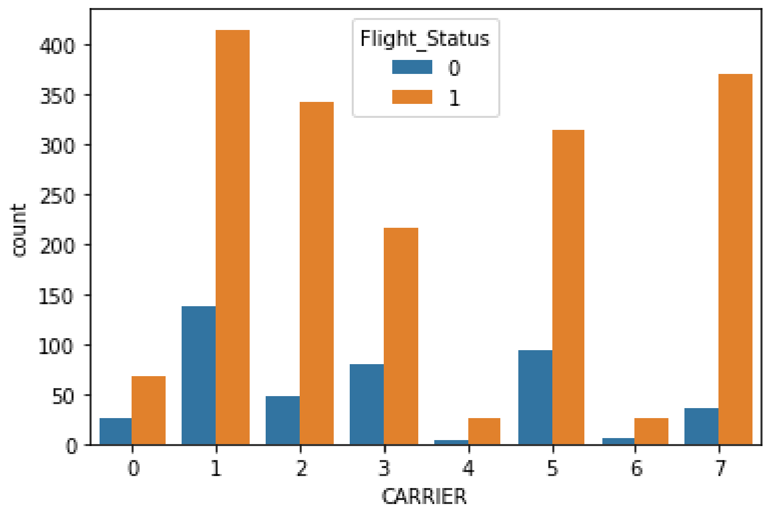


In [57]:
```
delayedflights = delays_dftemp.loc[delays_dftemp['Flight Status'] == "ontime"]
histplt = sns.histplot(data=delayedflights, x="DAY_WEEK", kde=True)
plt.show()
```



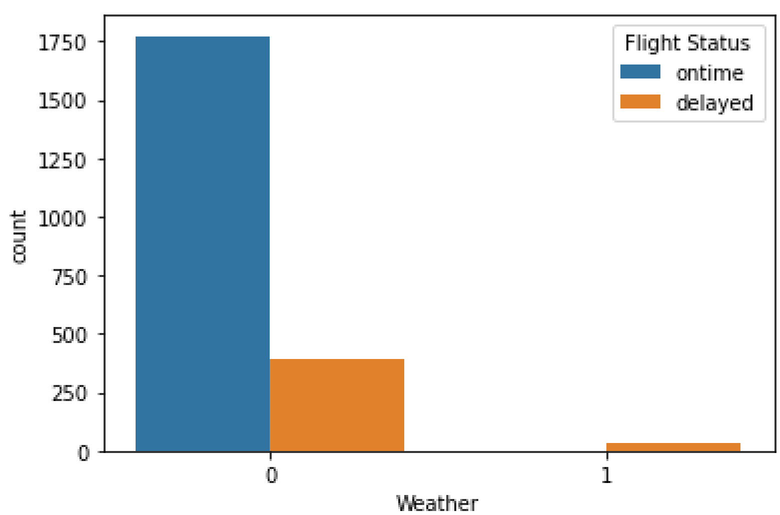
In [58]:
```
sns.countplot(data=delays_df, x="CARRIER", hue="Flight_Status")
```

Out[58]:
```
<AxesSubplot:xlabel='CARRIER', ylabel='count'>
```



In [59]:
```
sns.countplot(data=delays_dftemp, x="Weather", hue="Flight Status")
```

Out[59]:
```
<AxesSubplot:xlabel='Weather', ylabel='count'>
```



In [60]:
```
Corr_Matrix = delays_df.corr()
```

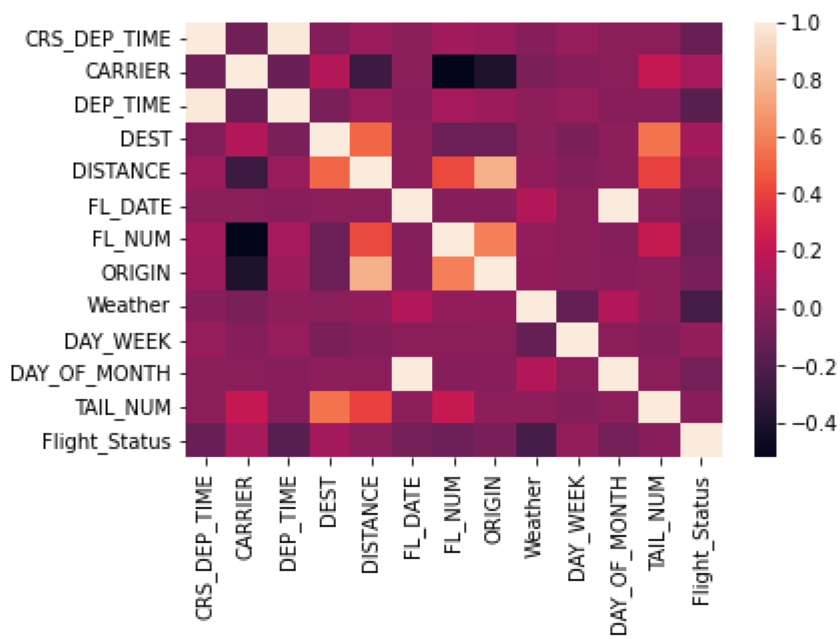
In [61]:
```
Corr_Matrix
```

Out[61]:

	CRS_DEP_TIME	CARRIER	DEP_TIME	DEST	DISTANCE	FL_DATE	FL_NU
CRS_DEP_TIME	1.000000	-0.082148	0.983523	-0.024689	0.062368	0.002635	0.0869
CARRIER	-0.082148	1.000000	-0.111835	0.149277	-0.284135	0.001608	-0.5223
DEP_TIME	0.983523	-0.111835	1.000000	-0.041426	0.057680	0.000446	0.1056
DEST	-0.024689	0.149277	-0.041426	1.000000	0.508091	0.018609	-0.1004
DISTANCE	0.062368	-0.284135	0.057680	0.508091	1.000000	0.006722	0.4219
FL_DATE	0.002635	0.001608	0.000446	0.018609	0.006722	1.000000	-0.0078
FL_NUM	0.086920	-0.522343	0.105660	-0.100415	0.421937	-0.007839	1.0000
ORIGIN	0.063465	-0.403164	0.070853	-0.103680	0.763863	-0.005592	0.5904
Weather	-0.008266	-0.043690	0.019001	0.004087	0.033104	0.143824	0.0420
DAY_WEEK	0.051766	-0.006251	0.051868	-0.045040	-0.020926	0.015994	0.0186
DAY_OF_MONTH	0.002324	0.001152	0.000132	0.019016	0.010121	0.997101	-0.0092
TAIL_NUM	0.013854	0.215656	0.000616	0.553189	0.393788	0.009735	0.2249
Flight_Status	-0.112474	0.102295	-0.170116	0.094228	0.018794	-0.065795	-0.1003

In [62]:
```
sns.heatmap(delays_df.corr())
```

Out[62]: <AxesSubplot:>



In [63]:
```
### separting correlation data for flight_status
corrdata = Corr_Matrix.Flight_Status
corrdata
```

Out[63]:
```
CRS_DEP_TIME    -0.112474
CARRIER          0.102295
DEP_TIME        -0.170116
DEST             0.094228
DISTANCE         0.018794
FL_DATE         -0.065795
FL_NUM          -0.100394
ORIGIN          -0.056666
Weather         -0.247217
DAY_WEEK         0.040756
DAY_OF_MONTH    -0.066598
TAIL_NUM        -0.002720
Flight_Status    1.000000
Name: Flight_Status, dtype: float64
```

using above table we can understand that: CARRIER, DEST, DAY_OF_MONTH are useful

using Predictive Power Score to dertermine best corelation with Flight Status

In [66]:
```
ppscoredf = pps.matrix(delays_df)
```

In [67]:
```
ppscoredf = ppscoredf.where(ppscoredf.x == 'Flight_Status','NA')
ppscoredf = ppscoredf[ppscoredf.x != 'NA']
ppscoredf = ppscoredf[ppscoredf.ppscore > 0]
```

In [68]:
```
ppscoredf
```

Out[68]:

	x	y	ppscore	case	is_valid_score	metric	baseline_score
156	Flight_Status	CRS_DEP_TIME	0.000516	regression	True	mean absolute error	361.144025
158	Flight_Status	DEP_TIME	0.010425	regression	True	mean absolute error	370.476602
168	Flight_Status	Flight_Status	1.0	predict_itself	True	None	0.0

above table states that CRS_DEP_TIME and DEP_TIME are correlated to Flight_Status

Now we create a new Dataframe after dropping unnecessary columns

In [71]:
```
maindf = delays_df[['CARRIER','DEST' ,'Weather', 'CRS_DEP_TIME', 'DEP_TIME', 'I
maindf.head()
```

Out[71]:

	CARRIER	DEST	Weather	CRS_DEP_TIME	DEP_TIME	DAY_OF_MONTH	Flight_Status
0	4	1	0	1455	1455	1	1
1	1	1	0	1640	1640	1	1
2	1	2	0	1245	1245	1	1
3	1	2	0	1715	1709	1	1
4	1	2	0	1039	1035	1	1

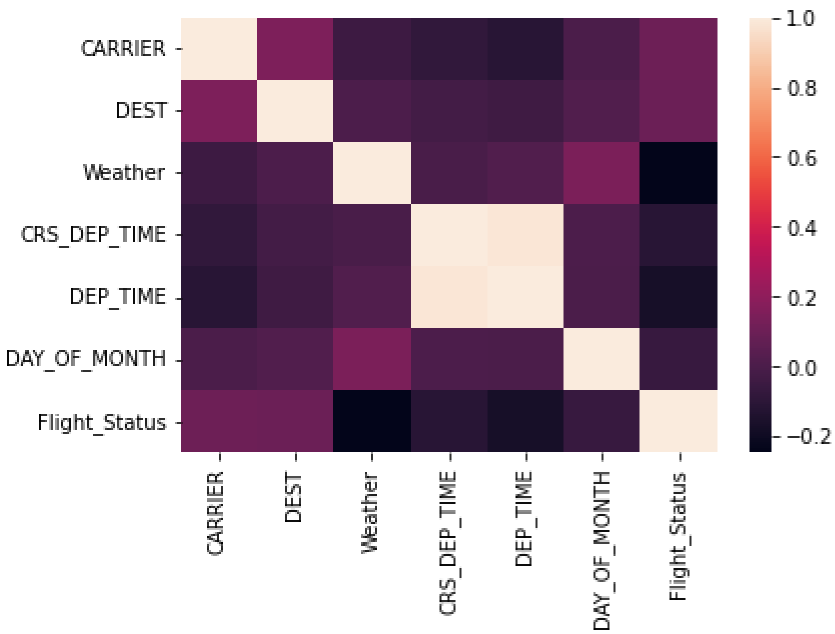
Saving the reduced data into CSV

In [107]:
```
maindf.to_csv('FlightDelaysTrainingData.csv')
```

Plotting heatmap to show correlation among selected variables with flight status

In [72]:
```
sns.heatmap(maindf.corr())
```

Out[72]: <AxesSubplot:>



Convert the DAY_WEEK column from numerical data into categorical data.

In [73]:
```

#This column will have 31 categories
delays_df.DAY_OF_MONTH = delays_df.DAY_OF_MONTH.astype('category')
```

In [75]:
```
#Create hourly bins departure time (original data has 100's of categories) so l
delays_df.CRS_DEP_TIME = [round(t / 100) for t in delays_df.CRS_DEP_TIME]
delays_df.CRS_DEP_TIME = delays_df.CRS_DEP_TIME.astype('category')
```

Naive Biase

Traing dataset with random shuffle 100 times and taking average of the accuracy for testing data

In [76]:
```
# list of all Naive Bayes accuracy outputs
naivAcc = []
```

In [77]:
```
for x in range(100):
    maindf = maindf.sample(frac = 1) #shuffling the dataframe

    #Split the data into training (80%) and testing (20%)
    predictors = ['CARRIER','DEST' ,'Weather', 'CRS_DEP_TIME', 'DEP_TIME', 'DA
    outcome = 'Flight_Status'
    X = pd.get_dummies(maindf[predictors])
    y = maindf['Flight_Status']
    classes = ['ontime', 'delayed']

    # split into training and validation
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.20

    # Run Naïve Bayes
    delays_nb = MultinomialNB(alpha=0.01)

    #Fit the model
    delays_nb.fit(X_train, y_train)

    #Predictions and accuracy
    y_train_pred = delays_nb.predict(X_train) # store the prediction data
    Temp = accuracy_score(y_train,y_train_pred) # calculate the accuracy

    #Appending the data to list
    naivAcc.append(Temp)
```

In [78]:
```
### average of 100 iterations of Naive Bays
np.average(naivAcc)
```

Out[78]:
```
0.8874659090909092
```

use model to make predictions on valid data

Model Diagnostics

In [80]:
```
# predict class membership (shows the class instead of probability by selecting
y_valid_pred = delays_nb.predict(X_valid)
y_train_pred = delays_nb.predict(X_train)
```

confusion matrix below

In [81]:
```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_valid,y_valid_pred)
cnf_matrix
```

Out[81]:
```
array([[ 31,  61],
       [  0, 349]], dtype=int64)
```

Logistic Regression Model

In [82]:
```
# list of all Naive Bayes accuracy outputs
LogAcc = []
```

Traing dataset with random shuffle 100 times and taking average of the accuracy for testing data

In [83]:
```
for x in range(100):
    maindf = maindf.sample(frac = 1) #shuffling the dataframe

    #Split the data into training (80%) and testing (20%)
    predictors = ['CARRIER','DEST' ,'Weather', 'CRS_DEP_TIME', 'DEP_TIME', 'DA
    outcome = 'Flight_Status'
    X = pd.get_dummies(maindf[predictors])
    y = maindf['Flight_Status']
    classes = ['ontime', 'delayed']

    # split into training and validation
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.20

    # Run Logistic Regression
    log_regression = LogisticRegression()

    #Fit the model
    log_regression.fit(X_train,y_train)

    #Predictions and accuracy
    y_valid_pred = log_regression.predict(X_valid)
    Temp = metrics.accuracy_score(y_valid,y_valid_pred)

    #Appending the data to list
    LogAcc.append(Temp)
```

In [84]:
```
### average of 100 iterations of Regression
np.average(LogAcc)
```

Out[84]:
```
0.8946258503401361
```

Model Diagnostics

In [85]:
```
#use model to make predictions on valid data
y_valid_pred = log_regression.predict(X_valid)
```

Confusion Matrix

In [86]:
```
#Once we fit the regression model, we can then analyze how well our model perfo
#First, we’ll create the confusion matrix for the model:
cnf_matrix = metrics.confusion_matrix(y_valid,y_valid_pred)
cnf_matrix
```

Out[86]:
```
array([[ 44,  37],
       [  3, 357]], dtype=int64)
```

CART algorithm

In [87]:
```
### Cart algorithm
cartAcc = []
cartdf = maindf
```

Traing dataset with random shuffle 100 times and taking average of the accuracy for testing data

In [88]:
```
for x in range(100):
    cartdf = cartdf.sample(frac = 1) #shuffling the dataframe

    #Split the data into training (80%) and testing (20%)
    #Independent variable set
    X=cartdf.iloc[:,0:6]
    #label set
    y=cartdf.iloc[:,6]

    # split into training and validation
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.20

    # Run Logistic Regression
    model=DecisionTreeClassifier(criterion='gini')

    #Fit the model
    model.fit(X_train,y_train)

    #Predictions and accuracy
    y_valid_pred = model.predict(X_valid)
    metrics.accuracy_score(y_valid,y_valid_pred)

    #Appending the data to list
    cartAcc.append(Temp)
```

In [89]:
```
### average of 100 iterations of CART
np.average(cartAcc)
```

Out[89]: 0.9092970521541952

Model Diagnostics

In [90]:
```
#use model to make predictions on valid data
y_valid_pred = model.predict(X_valid)
```

Confusion matrix

In [91]:
```
#Once we fit the regression model, we can then analyze how well our model perf
#First, we'll create the confusion matrix for the model:
cnf_matrix = metrics.confusion_matrix(y_valid,y_valid_pred)
cnf_matrix
```

Out[91]:
```
array([[ 47,  32],
       [ 33, 329]], dtype=int64)
```

Generating five new rows to predict flight status using CART algorithm

In [92]:
```
# Read the data file
test_df = pd.read_csv('FlightDelaysTestingData.csv')
```

In [93]:
```
# Datapreprocessing
l1=LabelEncoder()
#perform label encoding on 'team' column
test_df['CARRIER'] = l1.fit_transform(test_df['CARRIER'])
test_df['DEST'] = l1.fit_transform(test_df['DEST'])
test_df['ORIGIN'] = l1.fit_transform(test_df['ORIGIN'])
test_df['TAIL_NUM'] = l1.fit_transform(test_df['TAIL_NUM'])
test_df['FL_DATE'] = l1.fit_transform(test_df['FL_DATE'])
test_df['Flight Status'] = l1.fit_transform(test_df['Flight Status'])
```

In [94]:
```
test_df.head()
```

Out[94]:

	CRS_DEP_TIME	CARRIER	DEP_TIME	DEST	DISTANCE	FL_DATE	FL_NUM	ORIGIN	Weath
0	700	0	800	0	199	2	806	0	
1	1300	0	1255	0	199	3	808	0	
2	1200	0	1255	0	199	1	808	0	
3	1700	0	1833	0	199	0	810	0	
4	1300	0	1350	0	199	2	808	0	

In [95]:
```
X_valid= test_df[['CARRIER','DEST' ,'Weather', 'CRS_DEP_TIME', 'DEP_TIME', 'DA
X_valid
```

Out[95]:

	CARRIER	DEST	Weather	CRS_DEP_TIME	DEP_TIME	DAY_OF_MONTH
0	0	0	0	700	800	4
1	0	0	0	1300	1255	5
2	0	0	0	1200	1255	3
3	0	0	1	1700	1833	2
4	0	0	0	1300	1350	4

In [96]:
```
y_valid_pred = model.predict(X_valid)
```

Prediction for the new generated values

In [97]:
```
y_valid_pred
```

Out[97]:
```
array([1, 1, 1, 0, 0])
```

Our Predictions for flight status are matching with the generated row data

In [ ]: