In [1]:
```python
#iports need packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sn
```

In [2]:
```python
#loads the data set and shows the shape of it
data_df=pd.read_csv("C:/Users/predi/Documents/GitHub/DSC630/Datasets/airline_2m.csv", encoding='latin-1')
data_df.shape
```

```
C:\Users\predi\AppData\Local\Temp\ipykernel_19772\2826587435.py:2: DtypeWarning: Columns (69,76,77,84) have m
ixed types. Specify dtype option on import or set low_memory=False.
  data_df=pd.read_csv("C:/Users/predi/Documents/GitHub/DSC630/Datasets/airline_2m.csv", encoding='latin-1')
```

Out[2]:
```
(2000000, 109)
```

Data Cleaning

In [3]:
```python
#strips white sapce and lowers cases all of the headers and text.
data_df.columns=data_df.columns.str.lower()
data_df.columns=data_df.columns.str.strip()
```

In [4]:
```python
#Checks for null values in the data set and shows all columns with missing values
columns_with_null = data_df.columns[data_df.isnull().any()]
columns_null_count = data_df[columns_with_null].isnull().sum()
columns_above_90na = columns_null_count[columns_null_count > 1800000]
```

In [5]:
```python
#Drops all columns missing more than 90% of their values
data_df.drop(columns=columns_above_90na.index, inplace=True)
data_df.shape
```

Out[5]:
```
(2000000, 61)
```

In [6]:
```python
#Rechecks for nulls now that largest offends have been removed
columns_with_null = data_df.columns[data_df.isnull().any()]
columns_null_count = data_df[columns_with_null].isnull().sum()
columns_null_count
```

Out[6]:
```
tail_number               391763
originstate                  646
originstatefips              646
originstatename              646
deststate                    594
deststatefips                594
deststatename                594
deptime                    36005
depdelay                   36068
depdelayminutes            36068
depdel15                   36068
departuredelaygroups       36068
taxiout                   415642
wheelsoff                 415677
wheelson                  417958
taxiin                    417847
arrtime                    39551
arrdelay                   41078
arrdelayminutes            41078
arrdel15                   41078
arrivaldelaygroups         41078
crselapsedtime               281
actualelapsedtime          41052
airtime                   419349
carrierdelay             1778197
weatherdelay             1778197
nasdelay                 1778197
securitydelay            1778197
lateaircraftdelay        1778197
divairportlandings       1253886
dtype: int64
```

In [7]:
```
#dropping all rows missing the dependant variable needed for the model
data_df.dropna(subset=['arrdelay'], inplace=True)
#dropping all rows with missing tail numbers due to these observations
#missing multiple other key data points
data_df.dropna(subset=['tail_number'], inplace=True)
#Rechecks for nulls now that some of the NaN's have been dropped
columns_with_null = data_df.columns[data_df.isnull().any()]
columns_null_count = data_df[columns_with_null].isnull().sum()
columns_null_count
```

Out[7]:
```
depdelay                      60
depdelayminutes               60
depdel15                      60
departuredelaygroups          60
wheelsoff                     34
wheelson                     111
airtime                        1
carrierdelay             1358816
weatherdelay             1358816
nasdelay                 1358816
securitydelay            1358816
lateaircraftdelay        1358816
divairportlandings        849361
dtype: int64
```

In [8]:
```
#Drops the NaN value rows for the columns with less than 60.
data_df.dropna(subset=[
    'arrdelay','wheelson','depdelay','wheelsoff','airtime'], inplace=True)
#Rechecks for nulls now that the tail number NaN's have been dropped
columns_with_null = data_df.columns[data_df.isnull().any()]
columns_null_count = data_df[columns_with_null].isnull().sum()
columns_null_count
```

Out[8]:
```
carrierdelay           1358640
weatherdelay           1358640
nasdelay               1358640
securitydelay          1358640
lateaircraftdelay      1358640
divairportlandings      849216
dtype: int64
```

In [9]:
```
#fills the NaN values for the remaining columns that are needed for the model with 0
data_df.fillna(value=0,inplace=True)
#Checks for any other NaN values in the dataframe
data_df.columns[data_df.isnull().any()]
```
Out[9]:
```
Index([], dtype='object')
```

In [10]:
```
#Dropping string columns that that have a string and a matching ID column
redundant_columns=['reporting_airline','iata_code_reporting_airline','origin',
           'origincityname','originstate','originstatename','dest',
          'destcityname','deststate','deststatename','distancegroup',
                     'flightdate']
data_df.drop(columns=redundant_columns, inplace=True)
#dropping columns that aren't useful(59 min block every plane has sceduled)
data_df.drop(columns=['deptimeblk','arrtimeblk'], inplace=True)
```

In [11]:
```
#Now dropping any columns that would not be known before take off
#This is to make the model actually useful after take off the delay is
#relatively easy to calulate.
future_columns=['deptime','depdelay','depdelayminutes','depdel15','departuredelaygroups',
                'taxiout','wheelsoff','wheelson','taxiin','arrtime','arrdelayminutes',
                'arrdel15','arrivaldelaygroups','cancelled','diverted','actualelapsedtime',
                 'airtime','divairportlandings']
data_df.drop(columns=future_columns, inplace=True)
#Shows the final shape of the data frame
data_df.shape
```
Out[11]:
```
(1580411, 29)
```

In [12]:
```
#Changes the delay times to zeros and ones
#The delay can be expected but the time would not be known until takeoff
data_df['carrierdelay'] = data_df['carrierdelay'].apply(lambda x: 1 if x > 0 else x)
data_df['weatherdelay'] = data_df['weatherdelay'].apply(lambda x: 1 if x > 0 else x)
data_df['nasdelay'] = data_df['nasdelay'].apply(lambda x: 1 if x > 0 else x)
data_df['securitydelay'] = data_df['securitydelay'].apply(lambda x: 1 if x > 0 else x)
data_df['lateaircraftdelay'] = data_df['lateaircraftdelay'].apply(lambda x: 1 if x > 0 else x)
```

In [13]:
```
data_df
```
Out[13]:

	year	quarter	month	dayofmonth	dayofweek	dot_id_reporting_airline	tail_number	flight_number_reporting_airline	o
0	1998	1	1	2	5	19386	N297US	675	
1	2009	2	5	28	4	20437	N946AT	671	
2	2013	2	6	29	6	20398	N665MQ	3297	
3	2010	3	8	31	2	19790	N6705Y	1806	
4	2006	1	1	15	7	20355	N504AU	465	
...	...	...	...	...	...	...	...	...	
1999995	2008	1	3	23	7	19393	N712SW	966	
1999996	1999	1	1	5	2	19704	N14308	529	
1999997	2003	4	11	14	5	20355	N528AU	1457	
1999998	2012	2	5	15	2	19393	N281WN	536	
1999999	2003	2	4	29	2	19977	N364UA	1241	

1580411 rows × 29 columns

Data Prep for Model

In [15]:
```python
#import needed packages
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

In [16]:
```python
#turns tail number into a numeric representation
label_encoder = LabelEncoder()
data_df['tail_number'] = label_encoder.fit_transform(data_df['tail_number'])
```

In [17]:
```python
#Creates feature and target variables
x = data_df.drop('arrdelay', axis=1)
y = data_df['arrdelay']
```

In [18]:
```python
#Splits the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

Model Building

In [34]:
```python
#import needed packages
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

In [27]:
```python
#Creates a pipeline with different models
pipeline = Pipeline([("model", XGBRegressor())]) #place holder since none dose not work
#Defines parameters for the grid search for both models
params = [{'model': [XGBRegressor()],
           'model__n_estimators': [100, 500],
           'model__learning_rate': [0.3, 0.1]},
          {'model': [CatBoostRegressor()],
           'model__iterations': [100, 500],
           'model__learning_rate': [0.3, 0.1]}]
```

In [29]:
```python
#Performs a grid search for all the models
grid_search = GridSearchCV(pipeline, param_grid=params, cv=3)
grid_search.fit(x_train, y_train)
```

Out[29]:

```
▸ GridSearchCV
▸ estimator: Pipeline
    ▸ XGBRegressor
```

In [30]:

```python
#Evaluatse the best parameters and scores
print("Best parameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

```
Best parameters: {'model': <catboost.core.CatBoostRegressor object at 0x0000025FBECFD4E0>, 'model__iteration
s': 500, 'model__learning_rate': 0.1}
Best score: 0.385955662554791
```

In [31]:

```python
#Evaluates on the test set
test_score = grid_search.score(x_test, y_test)
print("Test Score:", test_score)
```

```
Test Score: 0.37458811810437853
```

In [32]:

```python
#Gets the best model from the grid search
best_model = grid_search.best_estimator_

#Predicts on the train and test sets
y_pred_train = best_model.predict(x_train)
y_pred = best_model.predict(x_test)
```

Result Interperation

In [35]:

```python
#Calculates the RMSE, MAE, and R2 on the train set
rmse_train = mean_squared_error(y_train, y_pred_train, squared=False)
mae_train = mean_absolute_error(y_train, y_pred_train)
r2_train = r2_score(y_train, y_pred_train)
#Calculates the RMSE, MAE, and R2 on the test set
rmse_test = mean_squared_error(y_test, y_pred, squared=False)
mae_test = mean_absolute_error(y_test, y_pred)
r2_test = r2_score(y_test, y_pred)
```

In [52]:

```python
print("average delay:", data_df['arrdelay'].mean(),"minutes")
print("------------Training data-------------")
print("train_RMSE:", rmse_train)
print("train_MAE:", mae_train)
print("train_R2:", r2_train)
print("------------testing data---------------")
print("test_RMSE:", rmse_test)
print("test_MAE:", mae_test)
print("test_R2:", r2_test)
```

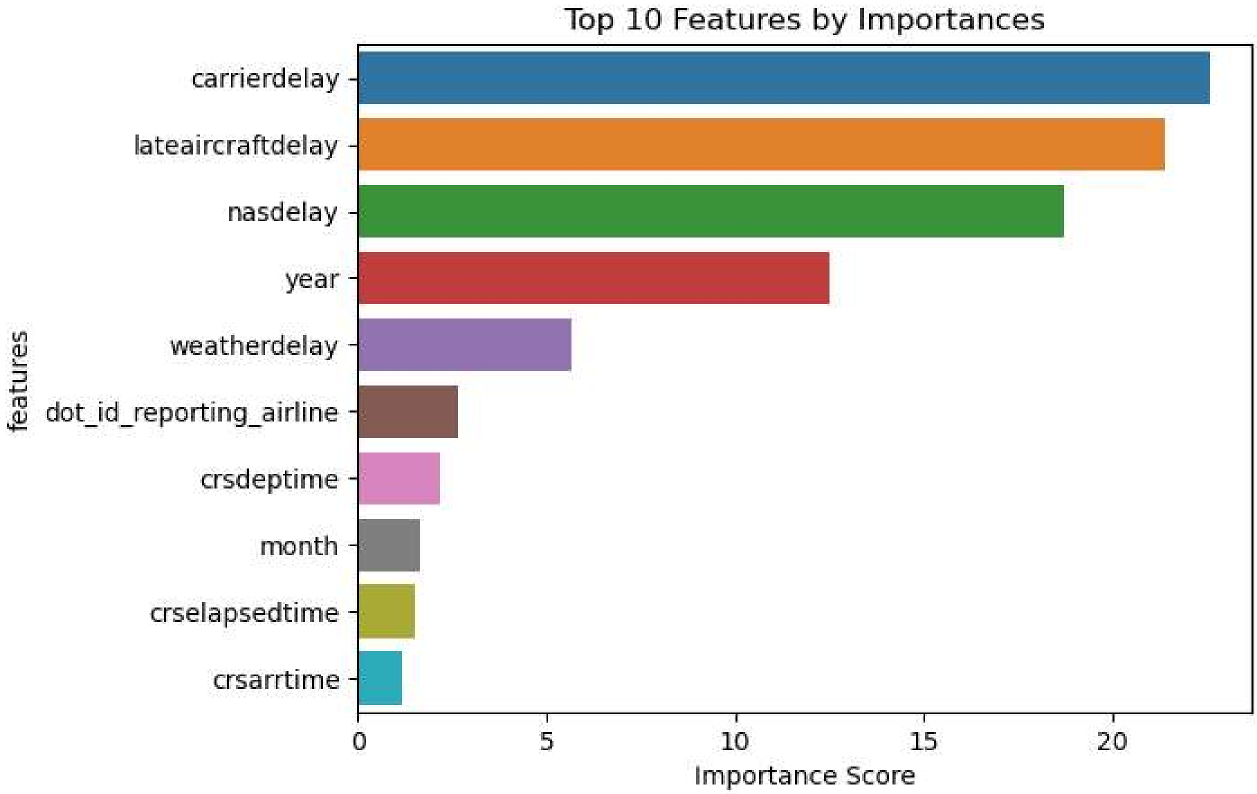
```
average delay: 6.20160831581152 minutes
------------Training data-------------
train_RMSE: 28.363256100141665
train_MAE: 14.45923410486165
train_R2: 0.4049634687084884
------------testing data---------------
test_RMSE: 29.997491435380784
test_MAE: 14.621067338119326
test_R2: 0.37458811810437853
```

In [56]:

```python
#Calculates the feature importance for the model
cat_model = best_model.named_steps['model']
feature_importance = pd.Series(cat_model.feature_importances_, index=x_train.columns)
feature_importance.sort_values(ascending=False, inplace=True)

#Creates a bar plot to display the top 10 features by importances
sn.barplot(x=feature_importance.head(10), y=feature_importance.head(10).index)
plt.xlabel('Importance Score')
plt.ylabel('features')
plt.title('Top 10 Features by Importances')
```

Out[56]: Text(0.5, 1.0, 'Top 10 Features by Importances')



In [ ]: