## Problem:

we need to predict the delay of flights and built a route recommendation using networkx.

## Dataset:

The U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, canceled, and diverted flights is published in DOT's monthly Air Travel Consumer Report and in this dataset of 2015 flight delays and cancellations.

The Dataset has #3 csv fils:

I have primarily used flights dataset.
It has 500 thousand rows and 31 columns.
The CSV file has daily chart of flights between two airports by various AIRLINE and between ORIGIN_AIRPORT and DESTINATION_AIRPORT along with ARRIVAL_DELAY and DEPARTURE_DELAY and other delays.

## Data Cleaning :

This picture is showing various missing values in columns and it”s filling factor.

I have removed the columns with filling factor less than 20 percent.

Also I after seeing that two columns CANCELLED and DIVERTED had all values only 0 these two were also removed to form a new dataset.

| | variable | missing values | filling factor (%) |
|---|---|---|---|
| 0 | CANCELLATION_REASON | 5729195 | 1.544643 |
| 1 | WEATHER_DELAY | 4755640 | 18.275040 |
| 2 | AIRLINE_DELAY | 4755640 | 18.275040 |
| 3 | SECURITY_DELAY | 4755640 | 18.275040 |
| 4 | AIR_SYSTEM_DELAY | 4755640 | 18.275040 |
| 5 | LATE_AIRCRAFT_DELAY | 4755640 | 18.275040 |
| 6 | ARRIVAL_DELAY | 105071 | 98.194371 |
| 7 | AIR_TIME | 105071 | 98.194371 |
| 8 | ELAPSED_TIME | 105071 | 98.194371 |
| 9 | TAXI_IN | 92513 | 98.410178 |
| 10 | WHEELS_ON | 92513 | 98.410178 |
| 11 | ARRIVAL_TIME | 92513 | 98.410178 |
| 12 | TAXI_OUT | 89047 | 98.469741 |
| 13 | WHEELS_OFF | 89047 | 98.469741 |
| 14 | DEPARTURE_TIME | 86153 | 98.519474 |
| 15 | DEPARTURE_DELAY | 86153 | 98.519474 |
| 16 | TAIL_NUMBER | 14721 | 99.747022 |
| 17 | SCHEDULED_TIME | 6 | 99.999897 |
| 18 | MONTH | 0 | 100.000000 |
| 19 | DAY | 0 | 100.000000 |
| 20 | DAY_OF_WEEK | 0 | 100.000000 |
| 21 | AIRLINE | 0 | 100.000000 |
| 22 | CANCELLED | 0 | 100.000000 |
| 23 | SCHEDULED_DEPARTURE | 0 | 100.000000 |
| 24 | FLIGHT_NUMBER | 0 | 100.000000 |
| 25 | SCHEDULED_ARRIVAL | 0 | 100.000000 |
| 26 | DESTINATION_AIRPORT | 0 | 100.000000 |
| 27 | DISTANCE | 0 | 100.000000 |
| 28 | ORIGIN_AIRPORT | 0 | 100.000000 |
| 29 | DIVERTED | 0 | 100.000000 |
| 30 | YEAR | 0 | 100.000000 |

```
flight.iloc[:,0:25].astype(bool).sum(axis=0
```

```
YEAR                 5714008
MONTH                5714008
DAY                  5714008
DAY_OF_WEEK          5714008
AIRLINE              5714008
FLIGHT_NUMBER        5714008
TAIL_NUMBER          5714008
ORIGIN_AIRPORT       5714008
DESTINATION_AIRPORT  5714008
SCHEDULED_DEPARTURE  5714008
DEPARTURE_TIME       5714008
DEPARTURE_DELAY      5385566
TAXI_OUT             5714008
WHEELS_OFF           5714008
SCHEDULED_TIME       5714008
ELAPSED_TIME         5714008
AIR_TIME             5714008
DISTANCE             5714008
WHEELS_ON            5714008
TAXI_IN              5714008
SCHEDULED_ARRIVAL    5714008
ARRIVAL_TIME         5714008
ARRIVAL_DELAY        5587795
DIVERTED                   0
CANCELLED                  0
dtype: int64
```

Different other rows which didn’t have much data of significance were also removed.
The Dataset had 24 columns then.

Now since 50 lakh were large no of rows to so I prepared another dataset of 10 lakh rows but which were in entirety of a single AIRLINE .Then I converted the added a new column of Date in datetime dtype.
And a columns of categorical data of object dtype were converted to categorical for faster operation.

## Feature Engineering :

Upon the columns of categorical data namely **'MONTH','AIRLINE','ORIGIN_AIRPORT', " DESTINATION_AIRPORT"** encoding .
Usually these features are represented by strings, and we need some way of transforming them to numbers before using scikit-learn's algorithms. The different ways of doing this are called *encodings*.

Now all the categorical columns I had were one with nominal values which means names in no order and no precedence whatsoever.

So I had options of going with

*One Way Encoding*

*Hash Encoding*

*Target Encoding*

*Catboost Encoding*

One Way Encoding was not working as expected and Hash Encoding was taking too long also there were too many ORIGIN_AIRPORT and DESTINATION_AIRPORT that Hash Encoding didn’t seem faesible due to increase in Dimensions .

I went with Target Encoding and Catboost Encoding of Categorical Datas using sklearn.pipeline .

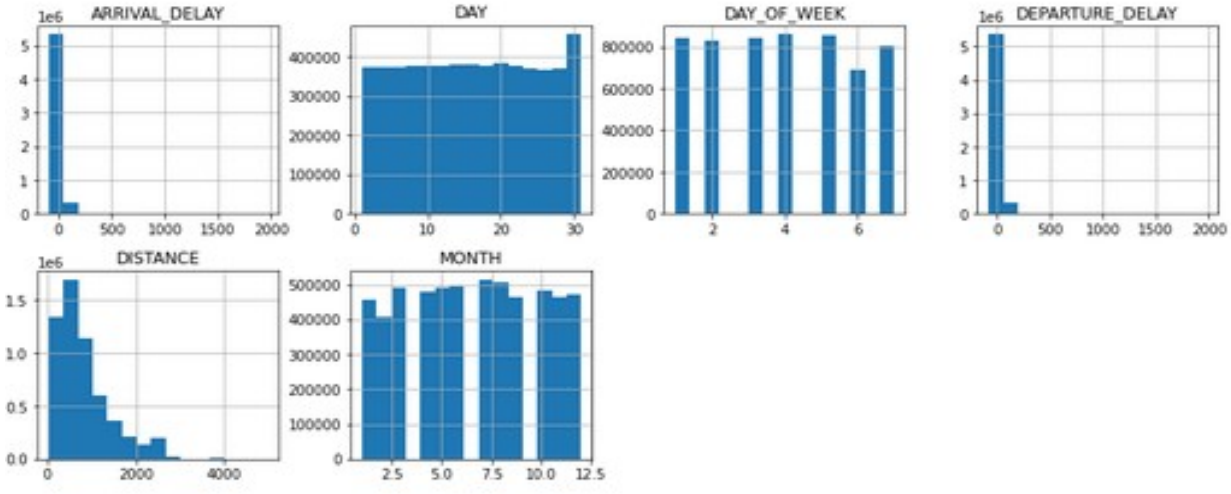
```
encoding_pipeline = Pipeline([

  ('encode_origin', ce.TargetEncoder(cols=['ORIGIN_AIRPORT'], return_
df=True)),
  ('encode_destination', ce.TargetEncoder(cols=['DESTINATION_AIRPORT
'], return_df=True)),
  ('encode_month', ce.TargetEncoder(cols=['MONTH'], return_df=True))
])

# Get the encoded dataset:
e_train = encoding_pipeline.fit_transform(train, train['ARRIVAL_DELAY
'])

e_test = encoding_pipeline.transform(test)
```

Model is fitted on train data and test data is transformed

## Experiments:

**Histogram:**

It shows the distribution of data in the numerical columns .



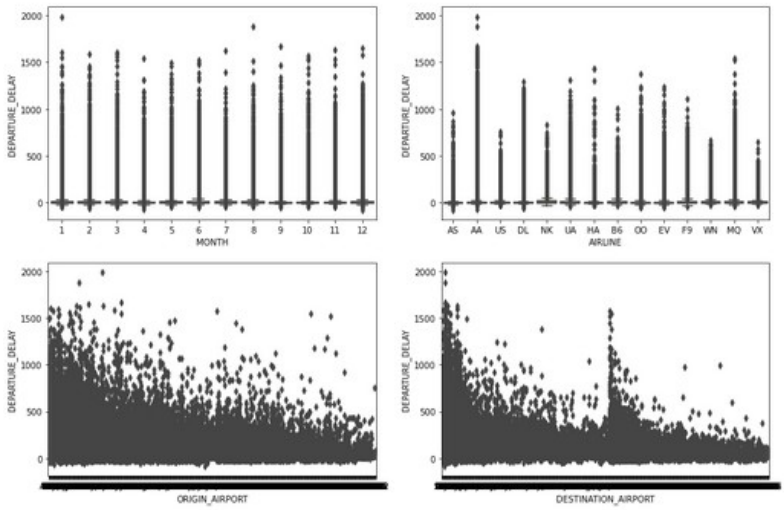

This shows that ARRIVAL_DELAY and DEPARTURE_DELAY are primarily very low and also negative which means that many time flights have arrived or departed before the scheduled time also.

There are however few outliers with 100 mins of ARRIVAL_DELAY and DEPARTURE_DELAY.

I also tried making **Boxplot** of the categorical data with the DEPARTURE_DELAY on the original dataset.

However due to large no of data point the experiment was not of much benefit.

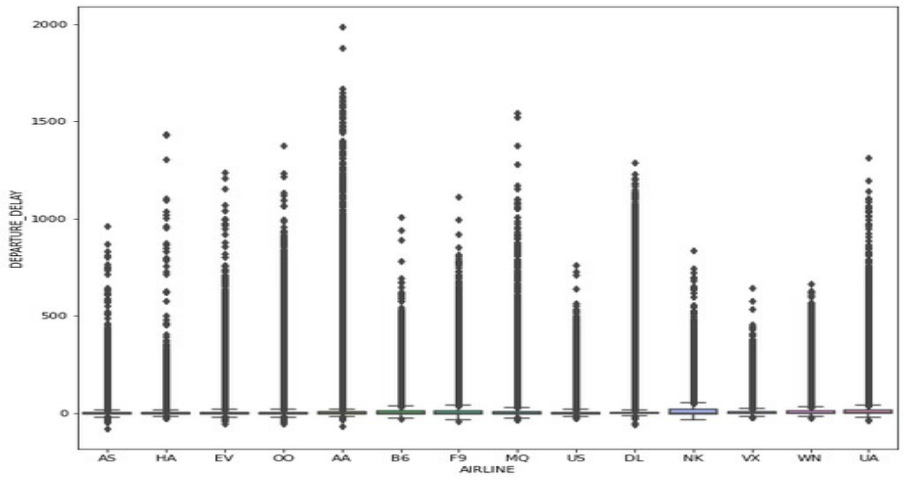
```
fig, ax = plt.subplots(2, 2, figsize=(15, 10))
for var, subplot in zip(categorical, ax.flatten()):
    sns.boxplot(x=var, y='DEPARTURE_DELAY', data=flight, ax=subplot)
```



**Boxplots** of different AIRLINES on with the DEPARTURE_DELAY. This shows that WN AIRLINE though having most no of AIRLINES around(10lakh) the range of DEPARTURE_DELAY is one of the least including the mean .

```
plt.figure(figsize=(10,10))
sorted_nb = flight.groupby(['AIRLINE'])['DEPARTURE_DELAY'].median().
sort_values()
sns.boxplot(x= flight['AIRLINE'], y= flight['DEPARTURE_DELAY'], orde
r=list(sorted_nb.index))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0ab3369da0>
```

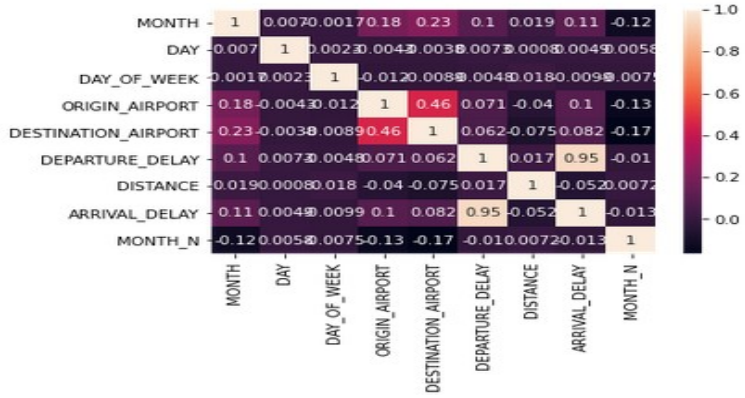



**Heatmap of Correlation matrix:**

It shows the relation of deprecated and encoded dataset of 10 lakh rows with its different columns.

```
sns.heatmap(e_train.corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f51a72aaeb8>
```



## Methods:

**Prediction:**

I used Cross Validation for Regression alogrithms :
When we're building a machine learning model using some data, we often split our data into training and validation/test sets. The training set is used to train the model, and the validation/test set is used to validate it on data it has never seen before. The classic approach is to do a simple 80%-20% split, sometimes with different values like 70%-30% or 90%-10%. In cross-validation, we do more than one split. We can do 3, 5, 10 or any K number of splits.

I used three methods to evaluate the best amongst:

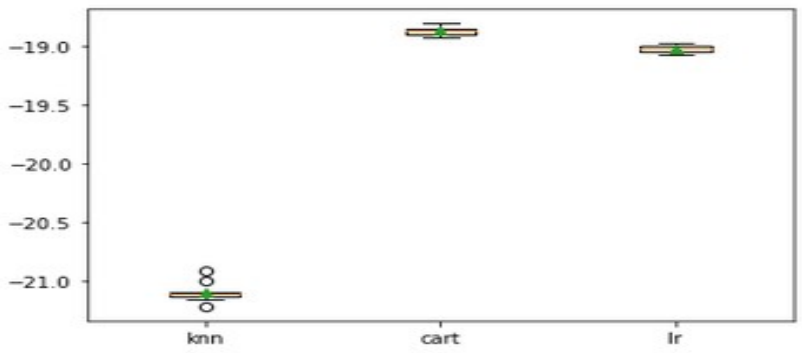
1)knn:KNeighborsRegressor(): KNN Regressor 2)cart:DecisionTreeRegressor()
3)lr:LinearRegression()

The models were evaluated with cross_val_score after 5 splits and 2 repetitions.
Then a boxplot of scores was made to see which method is best suited.

```
>knn -21.099 (0.078)
>cart -18.867 (0.035)
>lr -19.022 (0.031)
```

```
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

Boxplot and the negative MAE(Mean Absolute Error) was found that Desicion Tree best to be used over KNN, Linear Regression.




However I alongside I also used **Ensemble and Stacking**:To see whether it predicts better.

**Ensembling and Stacking**:Stacking is an ensemble machine learning algorithm that learns how to best combine the predictions from multiple well-performing machine learning models. The scikit-learn Python machine learning library provides an implementation of stacking for machine learning.

Using the model requires that you specify a list of estimators (level-0 models), and a final estimator (level-1 or meta-model).

A list of level-0 models or base models is provided via the "*estimators*" argument. This is

a Python list where each element in the list is a tuple with the name of the model

The dataset for the meta-model is prepared using cross-validation. By default, 5-fold cross-validation is used.

## Results of Prediction:

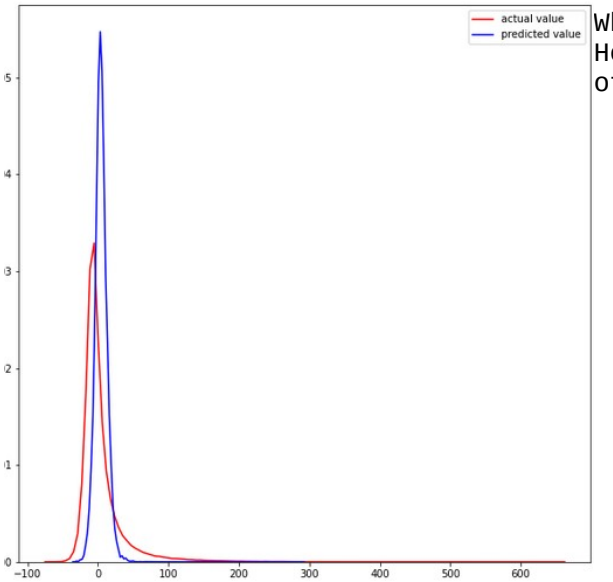
### Without Ensemble and Stacking:

1)
Columns :
**[['MONTH','ORIGIN_AIRPORT','DESTINATION_AIRPORT','DISTANCE']]**

Encoding : Target Encoding

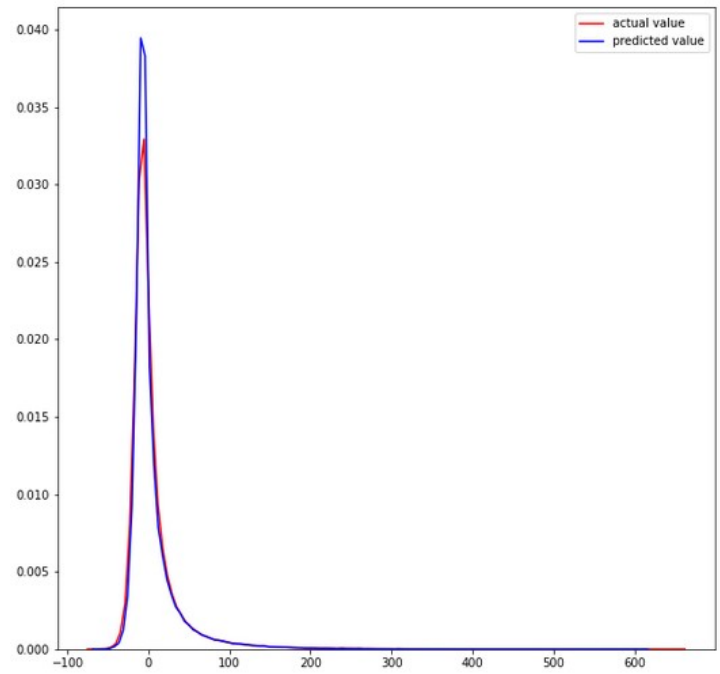
With the cross-validation of 5: On DecisionTreeRegressor()

The distplot graph of test data and predicted data is :




Which is somewhat good but not upto the mark .
However we were able to predict ARRIVAL_DELAY of AIRLINE (WN) by not so related factors .

2) Extra feature of DEPARTURE_DELAY is added.
Columns :
**[['MONTH','ORIGIN_AIRPORT','DESTINATION_AIRPORT','DISTANCE','DEPARTURE_DELAY']]**




Encoding : Target Encoding

With the cross-validation of 5: On DecisionTreeRegressor()

The distplot graph of test data and predicted data is :

Which is satisfyingly good and accurate.

This method is finally used for prediction even over stacking and ensemble

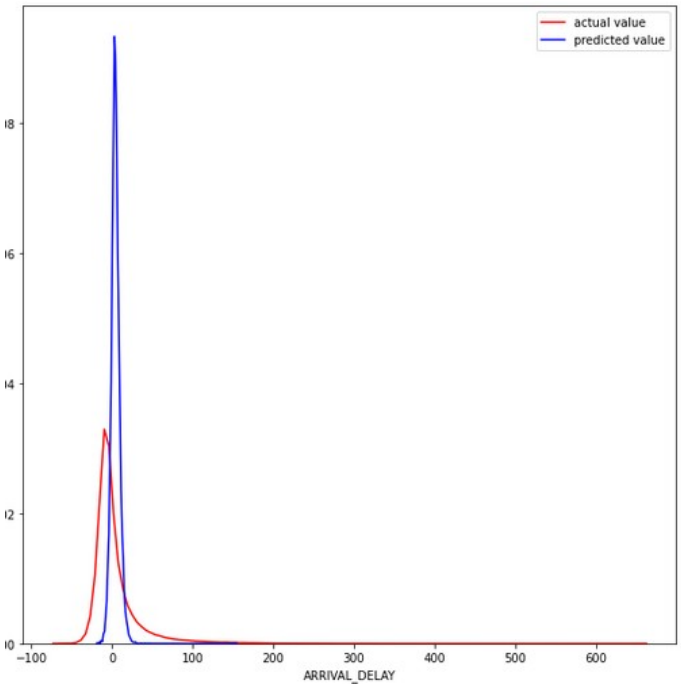
With Ensemble and Stacking :

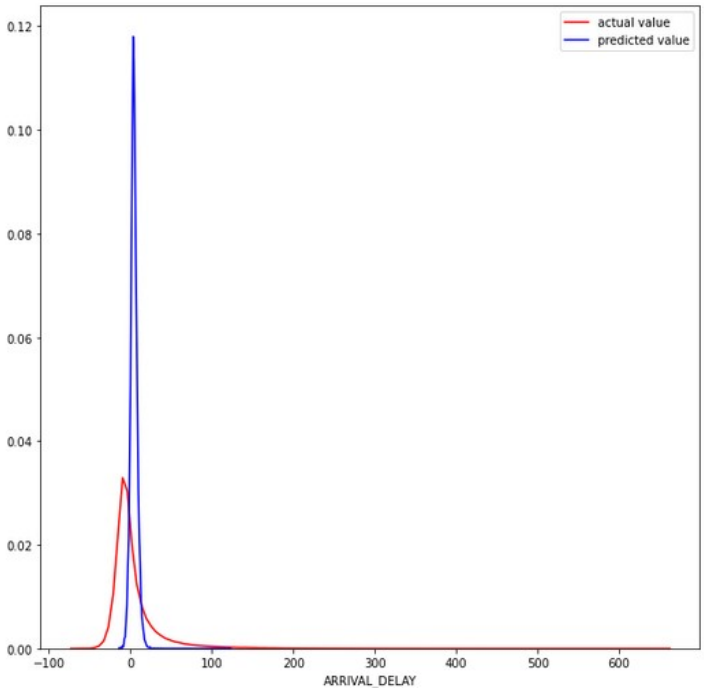
1)
level0 : KneighborsRegressor() and DecisionTreeRegressor()

level1 = LinearRegression()

Columns :
**[['MONTH','ORIGIN_AIRPORT','DESTINATION_AIRPORT','DISTANCE']]**

Encoding : CatBoost Encoding and Target Encoding used simultaneously:




In the both the cases results were very poor probably due to over-fitting.

Models were used by changing level0 and level1 of stacking to knn and decision tree respectively but model was equally over fitted.

2) With Extra feature of DEPARTURE_DELAY added.

level0 : KneighborsRegressor() and DecisionTreeRegressor()

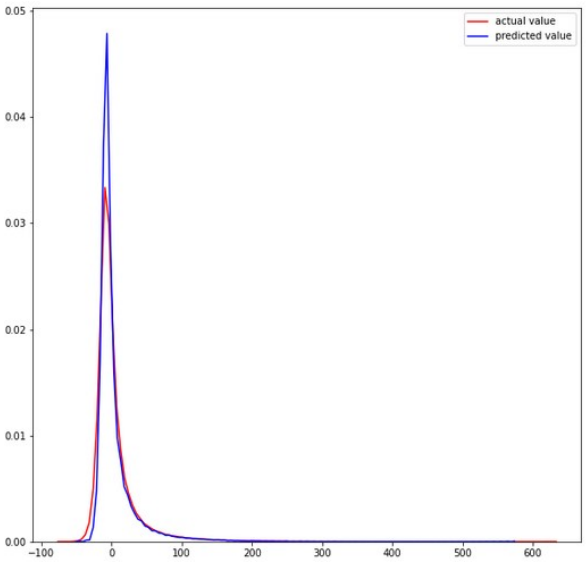
level1 = LinearRegression()

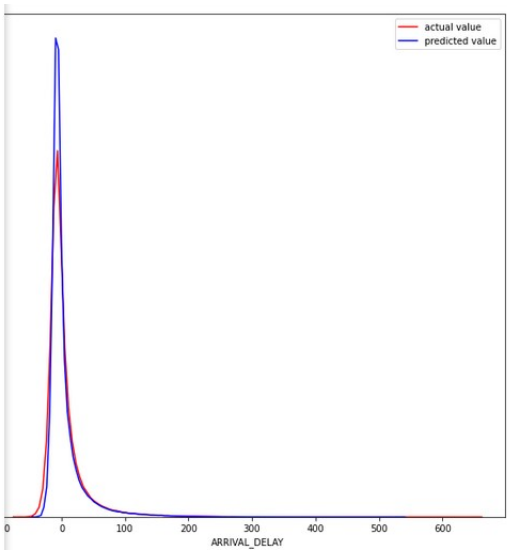
Columns :

**[['MONTH','ORIGIN_AIRPORT','DESTINATION_AIRPORT','DISTANCE','DEPARTURE_DELAY']]**

Encoding : CatBoost Encoding and Target Encoding used simultaneously:

Results were better and satisfactory but for final prediction I went with DecisionTreeRegressor() only with no Ensemble and Stacking.




## NetworkX:

A weighted network graph was created with Edges as 'ORIGIN_AIRPORT', " DESTINATION_AIRPORT" and weight added was predicted using DecisionTreeRegressor() with cross-validation of splits=5 which is Predicted_delay_cv(Predicted delay using cross-validation)

First of all to account for the early arrivals of airlines i.e negative DEPARTURE_DELAY and ARRIVAL_DELAY . The min value of delay as -66 . 70 was added to entire Predicted_delay_cv column.

And then shortest route with least delay between two airports was found :

```
nx.single_source_dijkstra(route_graph,  'ONT', 'STL')
```

```
(115.0, ['ONT', 'MDW', 'STL'])
```

The ARRIVAL_DELAY of 115 which is actually (115-140) .
Which means flight was 25 min earlier.

And then shortest route with least distance between two airports was found :

```
nx.single_source_dijkstra(route_dis_graph,  'ONT', 'STL')
```

Both the routes are different .

```
(1568, ['ONT', 'LAS', 'STL'])
```