First will start by importing the libraries that we need Pandas, Numpy, matplotlib & seaborn

In [1]:
```
import pandas as pd
import numpy as np
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
from pylab import rcParams
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from datetime import date
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

Then we need to read our data

In [46]:
```
df = pd.read_csv("C:\\Users\\mrnaz\\Desktop\\BootCamp\\My_Project\\WN-JAN-2020.csv")
```

let's have a look at what just we read

In [3]:
```
df.shape
```

Out[3]: (109770, 36)

In [4]:
```
df.head(5)
```

Out[4]:

	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	MKT_CARRIER_FL_NUM	TAIL_NUM	ORIGIN	ORIGIN_CITY_NAME	ORIGIN_STATE_NM	DEST	DEST_CITY_N
0	1	3	1/1/2020	5888	N951WN	ONT	Ontario, CA	California	SFO	San Francisc
1	1	3	1/1/2020	6276	N467WN	ONT	Ontario, CA	California	SFO	San Francisc
2	1	3	1/1/2020	4598	N7885A	ONT	Ontario, CA	California	SJC	San Jos
3	1	3	1/1/2020	4761	N551WN	ONT	Ontario, CA	California	SJC	San Jos
4	1	3	1/1/2020	5162	N968WN	ONT	Ontario, CA	California	SJC	San Jos

5 rows × 36 columns

First I will rename some columns

In [47]:
```
df.rename({'MKT_CARRIER_FL_NUM':'FL_NUM',
           'ORIGIN_CITY_NAME':'ORIGIN_CITY',
           'ORIGIN_CITY_NAME':'ORIGIN_CITY',
           'ORIGIN_STATE_NM':'ORIGIN_STATE',
           'DEST_CITY_NAME':'DEST_CITY',
          'DAY_OF_MONTH':'MONTH_DAY',
           'DAY_OF_WEEK':'WEEK_DAY',
           'DEST_STATE_NM':'DEST_STATE'},
          axis=1, inplace=True)
```

Second I will remove the Time in Hourly Block and the 3 letters city code it is not familiar for most people that are not working in the aviation let's just remove it and save some space

In [48]:
```
df.drop(['ORIGIN','DEST','DEP_TIME_BLK','ARR_TIME_BLK'],axis=1,inplace=True)
```

we decided that we will not include canceled flights in our study so we will remove the row that have tha value of 1 in the column CANCELLED then we will remove the columns CANCELLED and CANCELLATION_CODE

In [49]:
```
df = df[df.CANCELLED == 0]
```

In [50]:
```
df.drop(['CANCELLED','CANCELLATION_CODE'],axis=1,inplace=True)
```

let's have a look on the table after we do this changes

In [9]:
```
df.head(100)# or df.sample(5)
```

Out[9]:

	MONTH_DAY	WEEK_DAY	FL_DATE	FL_NUM	TAIL_NUM	ORIGIN_CITY	ORIGIN_STATE	DEST_CITY	DEST_STATE	CRS_DEP_TIME	...	ARR_DELAY_
0	1	3	1/1/2020	5888	N951WN	Ontario, CA	California	San Francisco, CA	California	1810	...	
1	1	3	1/1/2020	6276	N467WN	Ontario, CA	California	San Francisco, CA	California	1150	...	
2	1	3	1/1/2020	4598	N7885A	Ontario, CA	California	San Jose, CA	California	2020	...	
3	1	3	1/1/2020	4761	N551WN	Ontario, CA	California	San Jose, CA	California	1340	...	
4	1	3	1/1/2020	5162	N968WN	Ontario, CA	California	San Jose, CA	California	915	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
96	1	3	1/1/2020	3239	N8309C	Phoenix, AZ	Arizona	Nashville, TN	Tennessee	700	...	
97	1	3	1/1/2020	3896	N8621A	Phoenix, AZ	Arizona	Nashville, TN	Tennessee	935	...	
98	1	3	1/1/2020	4833	N707SA	Phoenix, AZ	Arizona	Nashville, TN	Tennessee	1930	...	
99	1	3	1/1/2020	5763	N774SW	Phoenix, AZ	Arizona	Nashville, TN	Tennessee	1520	...	
100	1	3	1/1/2020	3689	N401WN	Phoenix, AZ	Arizona	Boise, ID	Idaho	1510	...	

100 rows × 30 columns

O.K we need to know more about our table such as number of non-null and data type for each column

In [10]:
```
df.corr()  # View the correlations
```

Out[10]:

	MONTH_DAY	WEEK_DAY	FL_NUM	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	DEP_DELAY_NEW	TAXI_OUT	WHEELS_OFF	WH
MONTH_DAY	1.000000	-0.048708	-0.146981	-0.002648	-0.007319	-0.069223	-0.059010	0.034801	-0.006397	
WEEK_DAY	-0.048708	1.000000	0.398408	0.004037	0.008941	0.078601	0.068550	0.008830	0.008758	
FL_NUM	-0.146981	0.398408	1.000000	-0.008782	-0.004616	0.046484	0.039345	0.004174	-0.005141	
CRS_DEP_TIME	-0.002648	0.004037	-0.008782	1.000000	0.994650	0.106595	0.106980	-0.107341	0.992802	
DEP_TIME	-0.007319	0.008941	-0.004616	0.994650	1.000000	0.170195	0.169777	-0.103121	0.997372	
DEP_DELAY	-0.069223	0.078601	0.046484	0.106595	0.170195	1.000000	0.993329	0.038600	0.165301	

DEP_DELAY_NEW	-0.059010	0.068550	0.039345	0.106980	0.169777	0.993329	1.000000	0.034444	0.164698
TAXI_OUT	0.034801	0.008830	0.004174	-0.107341	-0.103121	0.038600	0.034444	1.000000	-0.083167
WHEELS_OFF	-0.006397	0.008758	-0.005141	0.992802	0.997372	0.165301	0.164698	-0.083167	1.000000
WHEELS_ON	0.013314	0.000322	-0.010252	0.772968	0.773501	0.080156	0.076225	-0.065948	0.775350
TAXI_IN	-0.020171	0.025409	0.023931	-0.018703	-0.018080	0.002004	0.001691	-0.002225	-0.017979
CRS_ARR_TIME	0.013295	-0.004613	-0.017346	0.755774	0.754418	0.094296	0.090490	-0.072465	0.753864
ARR_TIME	0.014994	-0.000259	-0.010940	0.756175	0.756619	0.076310	0.072296	-0.064768	0.758476
ARR_DELAY	-0.095738	0.083934	0.051736	0.099208	0.155911	0.894162	0.888264	0.230819	0.155496
ARR_DELAY_NEW	-0.056968	0.064739	0.036302	0.078027	0.136258	0.930269	0.942908	0.163794	0.133792
CRS_ELAPSED_TIME	-0.007621	0.024424	0.015153	-0.082036	-0.075473	0.060622	0.044740	0.079996	-0.074128
ACTUAL_ELAPSED_TIME	-0.021888	0.030711	0.019712	-0.081613	-0.074904	0.062771	0.046639	0.165157	-0.071846
AIR_TIME	-0.024825	0.028719	0.018090	-0.070213	-0.063881	0.059572	0.043609	0.058554	-0.062965
DISTANCE	-0.015939	0.029239	0.017418	-0.068282	-0.061814	0.061599	0.045281	0.044667	-0.061316
CARRIER_DELAY	0.023220	-0.011529	-0.012258	-0.064064	0.027886	0.558424	0.558531	-0.105624	0.015079
WEATHER_DELAY	0.018947	-0.001369	-0.020280	-0.043834	-0.013415	0.164455	0.164830	0.035696	-0.011454
NAS_DELAY	0.017258	0.008479	-0.001053	-0.153495	-0.153051	-0.048814	-0.044865	0.402928	-0.133158
SECURITY_DELAY	-0.018426	0.015569	0.011928	0.000769	0.002944	0.007443	0.007273	-0.012294	0.002516
LATE_AIRCRAFT_DELAY	0.001895	0.028032	0.019426	0.143588	0.209636	0.615008	0.615442	-0.106225	0.186857

24 rows × 24 columns

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 107849 entries, 0 to 109769
Data columns (total 30 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   MONTH_DAY            107849 non-null  int64  
 1   WEEK_DAY             107849 non-null  int64  
 2   FL_DATE              107849 non-null  object 
 3   FL_NUM               107849 non-null  int64  
 4   TAIL_NUM             107849 non-null  object 
 5   ORIGIN_CITY          107849 non-null  object 
 6   ORIGIN_STATE         107849 non-null  object 
 7   DEST_CITY            107849 non-null  object 
 8   DEST_STATE           107849 non-null  object 
 9   CRS_DEP_TIME         107849 non-null  int64  
 10  DEP_TIME             107849 non-null  float64
 11  DEP_DELAY            107849 non-null  float64
 12  DEP_DELAY_NEW        107849 non-null  float64
 13  TAXI_OUT             107849 non-null  float64
 14  WHEELS_OFF           107849 non-null  float64
 15  WHEELS_ON            107814 non-null  float64
 16  TAXI_IN              107814 non-null  float64
 17  CRS_ARR_TIME         107849 non-null  int64  
 18  ARR_TIME             107814 non-null  float64
 19  ARR_DELAY            107708 non-null  float64
 20  ARR_DELAY_NEW        107708 non-null  float64
 21  CRS_ELAPSED_TIME     107849 non-null  int64  
 22  ACTUAL_ELAPSED_TIME  107708 non-null  float64
 23  AIR_TIME             107708 non-null  float64
 24  DISTANCE             107849 non-null  int64  
 25  CARRIER_DELAY        10321 non-null   float64
 26  WEATHER_DELAY        10321 non-null   float64
 27  NAS_DELAY            10321 non-null   float64
 28  SECURITY_DELAY       10321 non-null   float64
 29  LATE_AIRCRAFT_DELAY  10321 non-null   float64
dtypes: float64(17), int64(7), object(6)
memory usage: 25.5+ MB
```

Let's convert the column FL_DATE from object to Date

In [51]:
```
df["FL_DATE"] = pd.to_datetime(df["FL_DATE"])
```

how still have nulls?

In [13]:
```
df.isna().any()
```

Out[13]:
```
MONTH_DAY             False
WEEK_DAY              False
FL_DATE               False
FL_NUM                False
TAIL_NUM              False
ORIGIN_CITY           False
ORIGIN_STATE          False
DEST_CITY             False
DEST_STATE            False
CRS_DEP_TIME          False
DEP_TIME              False
DEP_DELAY             False
DEP_DELAY_NEW         False
TAXI_OUT              False
WHEELS_OFF            False
WHEELS_ON              True
TAXI_IN                True
CRS_ARR_TIME          False
ARR_TIME               True
ARR_DELAY              True
ARR_DELAY_NEW          True
CRS_ELAPSED_TIME      False
ACTUAL_ELAPSED_TIME    True
AIR_TIME               True
DISTANCE              False
CARRIER_DELAY          True
WEATHER_DELAY          True
NAS_DELAY              True
SECURITY_DELAY         True
LATE_AIRCRAFT_DELAY    True
dtype: bool
```

we need to know number of NaN in each colomn

In [14]:
```
df.isna().sum()  # or df.isnull().sum()
```

```
DEP_DELAY_NEW             0
TAXI_OUT                  0
WHEELS_OFF                0
WHEELS_ON                35
TAXI_IN                  35
CRS_ARR_TIME              0
ARR_TIME                 35
ARR_DELAY               141
ARR_DELAY_NEW           141
CRS_ELAPSED_TIME          0
ACTUAL_ELAPSED_TIME     141
AIR_TIME                141
DISTANCE                  0
CARRIER_DELAY         97528
WEATHER_DELAY         97528
NAS_DELAY             97528
SECURITY_DELAY        97528
```

```
LATE_AIRCRAFT_DELAY    97528
dtype: int64
```

In the columns CARRIER_DELAY, WEATHER_DELAY, NAS_DELAY, SECURITY_DELAY and LATE_AIRCRAFT_DELAY The null means that there is no delay we will just replace the nulls with 0

In [52]:
```
#replac null with 0
df['CARRIER_DELAY'] = df['CARRIER_DELAY'].fillna(0)
df['WEATHER_DELAY'] = df['WEATHER_DELAY'].fillna(0)
df['NAS_DELAY'] = df['NAS_DELAY'].fillna(0)
df['SECURITY_DELAY'] = df['SECURITY_DELAY'].fillna(0)
df['LATE_AIRCRAFT_DELAY'] = df['LATE_AIRCRAFT_DELAY'].fillna(0)
```

We found nulls in columns WHEELS_ON, TAXI_IN, ARR_TIME, ACTUAL_ELAPSED_TIME and AIR_TIME

But this columns should not have nulls so we will consider it un complete data and we will exclude it from the study

In [53]:
```
#remove raw with null
df = df[df.WHEELS_ON.notnull()]
df = df[df.TAXI_IN.notnull()]
df = df[df.ARR_TIME.notnull()]
df = df[df.ACTUAL_ELAPSED_TIME.notnull()]
df = df[df.AIR_TIME.notnull()]
```

more descriptions

In [17]:
```
df.describe()
```

Out[17]:

	MONTH_DAY	WEEK_DAY	FL_NUM	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	DEP_DELAY_NEW	TAXI_OUT	WHEELS_OFF	WI
count	107708.000000	107708.000000	107708.000000	107708.000000	107708.000000	107708.000000	107708.000000	107708.000000	107708.000000	107
mean	15.977875	3.928613	2250.675864	1337.005144	1341.719882	4.321824	6.595341	12.020073	1361.953272	1
std	9.031045	1.913497	1707.669940	479.038444	485.280712	21.071903	20.188246	5.975961	483.994386	
min	1.000000	1.000000	1.000000	500.000000	1.000000	-20.000000	0.000000	1.000000	1.000000	
25%	8.000000	2.000000	971.000000	915.000000	920.000000	-4.000000	0.000000	9.000000	932.000000	1
50%	16.000000	4.000000	1749.000000	1330.000000	1336.000000	-1.000000	0.000000	10.000000	1347.000000	1
75%	24.000000	5.000000	3260.250000	1740.000000	1746.000000	4.000000	4.000000	14.000000	1758.000000	1
max	31.000000	7.000000	6860.000000	2335.000000	2359.000000	602.000000	602.000000	115.000000	2400.000000	2

8 rows × 24 columns

To examine data types and look for cases of missing or potentially wrong data, the describe function helps to get an overview of the data from a statistical summary point of view. This is also useful for spotting any potential errors that may need a closer look.

In [18]:
```
df.shape
```

Out[18]: (107708, 30)

There are 109770 records and 36 columns in the dataset.

In [19]:
```
df.size
```

Out[19]: 3231240

In [20]:
```
df.columns
```

Out[20]:
```
Index(['MONTH_DAY', 'WEEK_DAY', 'FL_DATE', 'FL_NUM', 'TAIL_NUM', 'ORIGIN_CITY',
       'ORIGIN_STATE', 'DEST_CITY', 'DEST_STATE', 'CRS_DEP_TIME', 'DEP_TIME',
       'DEP_DELAY', 'DEP_DELAY_NEW', 'TAXI_OUT', 'WHEELS_OFF', 'WHEELS_ON',
       'TAXI_IN', 'CRS_ARR_TIME', 'ARR_TIME', 'ARR_DELAY', 'ARR_DELAY_NEW',
       'CRS_ELAPSED_TIME', 'ACTUAL_ELAPSED_TIME', 'AIR_TIME', 'DISTANCE',
       'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY',
       'LATE_AIRCRAFT_DELAY'],
      dtype='object')
```

In [21]:
```
df.dtypes # This will peovied data type for each column
```

```
DEP_DELAY_NEW              float64
TAXI_OUT                   float64
WHEELS_OFF                 float64
WHEELS_ON                  float64
TAXI_IN                    float64
CRS_ARR_TIME                 int64
ARR_TIME                   float64
ARR_DELAY                  float64
ARR_DELAY_NEW              float64
CRS_ELAPSED_TIME             int64
ACTUAL_ELAPSED_TIME        float64
AIR_TIME                   float64
DISTANCE                     int64
CARRIER_DELAY              float64
WEATHER_DELAY              float64
NAS_DELAY                  float64
SECURITY_DELAY             float64
LATE_AIRCRAFT_DELAY        float64
dtype: object
```

In [22]:
```
# i need to list all origin city name
print(df.ORIGIN_CITY.unique())
```

```
['Ontario, CA' 'Norfolk, VA' 'West Palm Beach/Palm Beach, FL'
 'Portland, OR' 'Philadelphia, PA' 'Phoenix, AZ' 'Pittsburgh, PA'
 'Pensacola, FL' 'Providence, RI' 'Portland, ME' 'Raleigh/Durham, NC'
 'Richmond, VA' 'Reno, NV' 'Rochester, NY' 'Fort Myers, FL'
 'San Diego, CA' 'San Antonio, TX' 'Louisville, KY' 'Seattle, WA'
 'San Francisco, CA' 'San Jose, CA' 'San Juan, PR' 'Salt Lake City, UT'
 'Sacramento, CA' 'Santa Ana, CA' 'St. Louis, MO' 'Tampa, FL' 'Tulsa, OK'
 'Tucson, AZ' 'Albuquerque, NM' 'Albany, NY' 'Amarillo, TX' 'Atlanta, GA'
 'Austin, TX' 'Dallas, TX' 'New Orleans, LA' 'Oakland, CA' 'Kahului, HI'
 'Oklahoma City, OK' 'Omaha, NE' 'Hartford, CT' 'Birmingham, AL'
 'Nashville, TN' 'Boise, ID' 'Boston, MA' 'Buffalo, NY' 'Burbank, CA'
 'Baltimore, MD' 'Charleston, SC' 'Cleveland, OH' 'Charlotte, NC'
 'Columbus, OH' 'Corpus Christi, TX' 'Cincinnati, OH' 'Washington, DC'
 'Denver, CO' 'Des Moines, IA' 'Detroit, MI' 'Panama City, FL'
 'El Paso, TX' 'Fort Lauderdale, FL' 'Spokane, WA' 'Grand Rapids, MI'
 'Greer, SC' 'Honolulu, HI' 'Houston, TX' 'Harlingen/San Benito, TX'
 'Wichita, KS' 'Indianapolis, IN' 'Islip, NY' 'Hilo, HI'
 'Jacksonville, FL' 'Kona, HI' 'Las Vegas, NV' 'Los Angeles, CA'
 'Lubbock, TX' 'New York, NY' 'Long Beach, CA' 'Lihue, HI'
 'Little Rock, AR' 'Midland/Odessa, TX' 'Kansas City, MO' 'Orlando, FL'
 'Chicago, IL' 'Memphis, TN' 'Manchester, NH' 'Milwaukee, WI'
 'Minneapolis, MN']
```

In [23]:
```
#To check if there is duplicate rows in the dataset
sum(df.duplicated())
```

Out[23]: 0

In [24]:
```
# This command tell us how many flights in each airport, it will basically calculate city name repeat number
print (df['ORIGIN_CITY'].value_counts())
```

```
Las Vegas, NV      5663
Dallas, TX         5635
```

```
Denver, CO          5564
Chicago, IL         5424
Baltimore, MD       5409
                    ...
Panama City, FL       97
Pensacola, FL         91
Portland, ME          90
Des Moines, IA        85
Hilo, HI              51
Name: ORIGIN_CITY, Length: 88, dtype: int64
```

In [25]:
```
#We need to how many filght for each plain
print (df.TAIL_NUM.value_counts())
```

```
N739GB    203
N942WN    202
N789SW    201
N938WN    201
N772QF    199
         ...
N7840A     26
N767SW     19
N8312C     14
N7838A      8
N7875A      6
Name: TAIL_NUM, Length: 704, dtype: int64
```

In [54]:
```
df[df['FL_NUM'] == 795]  # This will desplay all flight number 795
```

Out[54]:

	MONTH_DAY	WEEK_DAY	FL_DATE	FL_NUM	TAIL_NUM	ORIGIN_CITY	ORIGIN_STATE	DEST_CITY	DEST_STATE	CRS_DEP_TIME	...	ARR_DI
1468	30	4	2020-01-30	795	N7831B	Oklahoma City, OK	Oklahoma	Washington, DC	Virginia	1040	...	
3562	31	5	2020-01-31	795	N246LV	Washington, DC	Virginia	Tampa, FL	Florida	1510	...	
4019	31	5	2020-01-31	795	N246LV	Houston, TX	Texas	Oklahoma City, OK	Oklahoma	840	...	
5111	31	5	2020-01-31	795	N246LV	Oklahoma City, OK	Oklahoma	Washington, DC	Virginia	1040	...	
6477	6	1	2020-01-06	795	N779SW	Washington, DC	Virginia	Tampa, FL	Florida	1510	...	
6931	6	1	2020-01-06	795	N779SW	Houston, TX	Texas	Oklahoma City, OK	Oklahoma	840	...	
8016	6	1	2020-01-06	795	N779SW	Oklahoma City, OK	Oklahoma	Washington, DC	Virginia	1040	...	

The average of the departial delay

In [55]:
```
av_DEP_DELAY = df.DEP_DELAY.mean()
print (av_DEP_DELAY)
```

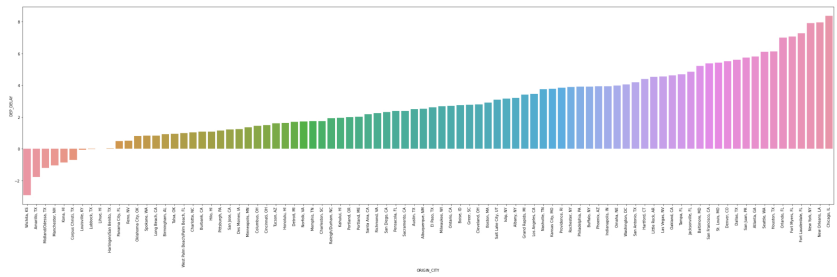
```
4.321823819957664
```

let's Display some Graphs

The below figure shows the delay departure time for each city, the minus number means early departure

In [28]:
```
plt.figure(figsize =(40,10))
plt.xticks(rotation = 90)
temp = df[['DEP_DELAY', 'ORIGIN_CITY']]
temp = pd.DataFrame(temp.groupby('ORIGIN_CITY')['DEP_DELAY'].agg('mean')).reset_index().sort_values(by= ['DEP_DELAY'])
temp
sns.barplot(x='ORIGIN_CITY', y="DEP_DELAY", data = temp)
```

Out[28]:
```
<AxesSubplot:xlabel='ORIGIN_CITY', ylabel='DEP_DELAY'>
```



In [29]:
```
# Here we can get the same information in a table
t1=temp
t1
```

Out[29]:

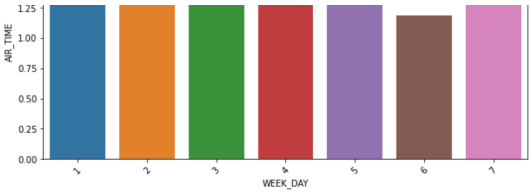
	ORIGIN_CITY	DEP_DELAY
87	Wichita, KS	-2.923729
2	Amarillo, TX	-1.783217
47	Midland/Odessa, TX	-1.205036
45	Manchester, NH	-1.042802
37	Kona, HI	-0.872483
...	...	...
24	Fort Myers, FL	7.084225
23	Fort Lauderdale, FL	7.290528
52	New York, NY	7.927954
51	New Orleans, LA	7.960164
13	Chicago, IL	8.377028

88 rows × 2 columns

The below figure shows the number of flights time in each day of the week (Air Traffic)

In [30]:
```
plt.figure(figsize =(10,5))
plt.xticks(rotation = 45)
temp = df[['AIR_TIME', 'WEEK_DAY']]
temp = pd.DataFrame(temp.groupby('WEEK_DAY')['AIR_TIME'].agg('sum')).reset_index().sort_values(by= ['AIR_TIME'])
temp
sns.barplot(x='WEEK_DAY', y="AIR_TIME", data = temp)
```

Out[30]:
```
<AxesSubplot:xlabel='WEEK_DAY', ylabel='AIR_TIME'>
```



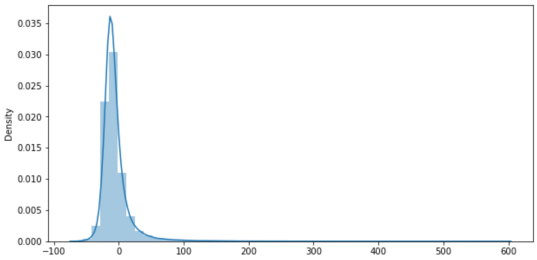
In [31]:
```
#Wednesday has the highest air traffic during the week
t2=temp
t2
```

Out[31]:

	WEEK_DAY	AIR_TIME
5	6	1182881.0
1	2	1440845.0
0	1	1443243.0
6	7	1478819.0
4	5	1768817.0
2	3	1822548.0
3	4	1855419.0

The rate of difference between the scheduled access time and the actual time

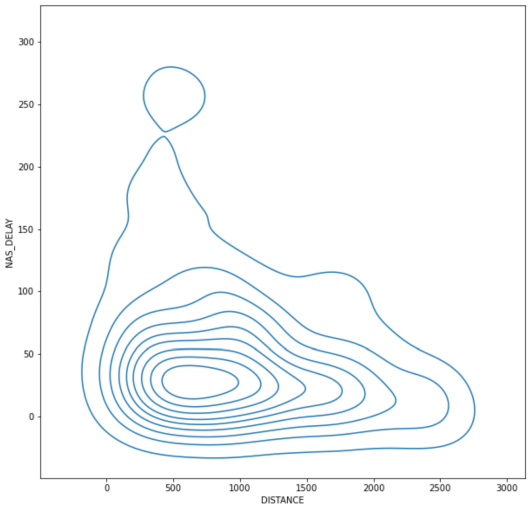
In [32]:
```
plt.figure(figsize = (10,5))
sns.distplot(x=df['ARR_DELAY']);
```



The figure below gives us an idea of the relationship between flight delays due to civil aviation and the distance between airports

In [33]:
```
plt.figure(figsize =(10,10))
temp = df[['NAS_DELAY', 'DISTANCE']]
temp = pd.DataFrame(temp.groupby('DISTANCE')['NAS_DELAY'].agg('max')).reset_index().sort_values(by= ['NAS_DELAY'])
temp
sns.kdeplot(x='DISTANCE', y="NAS_DELAY", data = temp)
```

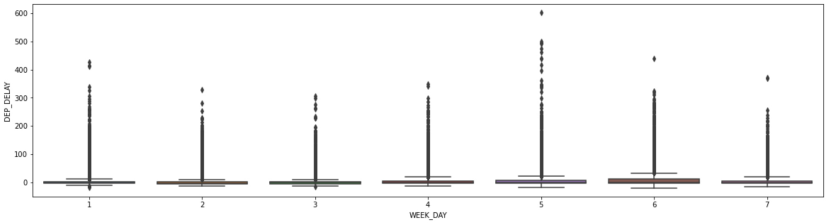
Out[33]: <AxesSubplot:xlabel='DISTANCE', ylabel='NAS_DELAY'>



the below graph shows the departial delay for each day of the week

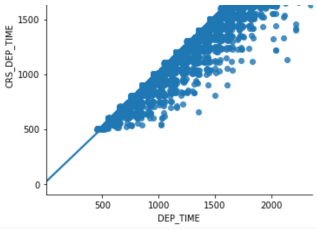
In [40]:
```
plt.figure(figsize =(20,5))
sns.boxplot(df.WEEK_DAY, df.DEP_DELAY)
```

Out[40]: <AxesSubplot:xlabel='WEEK_DAY', ylabel='DEP_DELAY'>



Modeling with sklearn to know the discipline in take-off times

In [87]:
```
sns.lmplot(x ='DEP_TIME', y ='CRS_DEP_TIME', data = df,ci=None );
```



In [90]:
```
# import model and fit
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression

# grab the DEP_TIME and CRS_DEP_TIME
#define the independent column
X1 = df.DEP_TIME.values.reshape(-1, 1)
#then the dependent column?
y1 = df.CRS_DEP_TIME.values.reshape(-1, 1)

#1 eate an object to linear Regression model formula
linreg_model = LinearRegression()

#2 tting the model == Train Model
linreg_model.fit(X1, y1)
```

Out[90]:
```
LinearRegression()
```

In [91]:
```
round(linreg_model.score(X1, y1)*100)
```

Out[91]:
```
99
```

In [92]:
```
import numpy as np
x2 = np.array(100)
x2 = x2.reshape(-1,1)

linreg_model.predict(x2)
```

Out[92]:
```
array([[117.81314602]])
```

In [ ]:
```

```