Project Report on Flight Delay Prediction Using PySpark

Purvaj Desai
A20469336

Parth Rathod
A20458817

Pinakin Nimavat
A20466027

Abstract

With the development of civil aviation, the number of flights keeps increasing and the flight delay has become a serious issue and even tends to normality. This paper aims to prove that Machine Learning algorithm has advantages in airport flight delay prediction. There are four supervised machine learning algorithms used including Decision Tree Algorithm, Random Forest, Logistic Regression, SVM. To verify the effectiveness of the proposed method, comparative experiments are carried out flight datasets from 2008 to 2018. We have made a comparative analysis based on Accuracy of each algorithm.

Introduction

The Bureau of Transportation Statistics of the United States Department of Transportation (DOT) monitors the on-time performance of domestic flights operated by significant airlines. The DOT's monthly Air Travel Consumer Report, as well as this dataset of 2009 - 2018; flight delays and cancellations, provide summary information on the number of on-time, delayed, canceled, and diverted flights. The need for air travel has risen considerably as the country's economy has grown rapidly. Flight delays are becoming increasingly severe, causing significant damage to the image of civil aviation services. For passengers, flight delays result in inconvenience, a bad mood, and a double loss of time and money; for the airport, flight delays have a significant impact on the normal operation of the airport; and for the airline, frequent flight delays not only result in significant financial losses, but also harm the airline's reputation. Flight delays have become a stumbling block to the aviation industry's growth.

In this research, we developed classification models to forecast whether the flight will be delayed at destination or not. We have selected three classes in which we will classify if the aircraft will arrive either "On time" or it will reach "early" or it will be "delayed" at the destination airport. The following is a list of the paper's research value: 1) Improving the accuracy of flight delay predictions and the ability of relevant departments to make decisions; 2) Promoting the use of computer simulation technology in civil aviation; 3) Assisting in the construction of civil aviation information.

Literature Review

Spark is a general-purpose distributed data processing engine that is suitable for use in a wide range of circumstances. On top of the Spark core data processing engine, there are libraries for SQL, machine learning, graph computation, and stream processing, which can be used together in an application. Programming languages supported by Spark include: Java, Python, Scala, and R. Application developers and data scientists incorporate Spark into their applications to rapidly query, analyze, and transform data at scale. Tasks most frequently associated with Spark include ETL and SQL batch jobs across large data sets, processing of streaming data from sensors, IoT, or financial systems, and machine learning tasks.

Spark is capable of handling several petabytes of data at a time, distributed across a cluster of thousands of cooperating physical or virtual servers. It has an extensive set of developer libraries and APIs and supports languages such as Java, Python, R, and Scala; its flexibility makes it well-suited for a range of use cases. Spark is often used with distributed data stores such as HPE Ezmeral Data Fabric, Hadoop's HDFS, and Amazon's S3, with popular NoSQL databases such as HPE Ezmeral Data Fabric, Apache HBase, Apache Cassandra, and MongoDB, and with distributed messaging stores such as HPE Ezmeral Data Fabric and Apache Kafka.

Project Objective and Our approach

Link for Dataset: [Airline_Flight_Delay_Dataset_2008-2018]

We are trying to accomplish a comparative analysis for hardware used on AWS. We started with deciding on the steps we have performed. After a long discussion and brainstorming we began with the exploration and analysis of data. As we have csv data files of each year i.e 2009,2010 until 2018; we performed the visualization on each dataset and on a combined dataset to better get an idea about how our data behaves and what are the key points that we had to think inorder to better understand the purpose. Based on the features selected we have created 4 Machine Learning Algorithms 1) Decision Tree; 2) Random Forest; 3) Logistic Regression; 4) SVM. We have **61,556,964 rows** and **28 features**. We have dropped rows with more than 80% of data being NULL. We have created Data Visualizations charts for better understanding of the data.We have manually created 10 features for the purpose of creating a Machine Learning Model. We are trying to classify into 3 Labels whether the Flight will be Delayed , Flight will be On TIme, Flight will be Early. We have used Multiclass Classification Metrics to show the Accuracy.

Project Configuration

AWS EMR	**Software**	**Instances (Master/Core Node)**	**Storage**
emr 6.4.0	**Spark 3.1.2**	**m4.xlarge (1 Master, 2 Cores)**	**S3 Bucket**
emr 6.4.0	**Spark 3.1.2**	**m4.xlarge (1 Master, 3 Cores)**	**S3 Bucket**
emr 6.4.0	**Spark 3.1.2**	**m4.xlarge (1 Master, 4 Cores)**	**S3 Bucket**

Feature Engineering

First we were trying to understand how many Null Values we had in each feature. This gives us the understanding of the features which are not important or the features that can't be used.

```python
def count_null(self,df):
    """
    :param df: Pass DataFrame whose Null values has to be counted
    :return: None
    """

    new_df = df.select([(((count(when(col(c).isNull(),c)))/df.count())*100).alias(c) for c in df.columns])
    new_df.show()
    print("\n")
```

```
Percentage of Null Values in 2018 Year for Each Column:
+-------+----------+-----------------+------+----+------------+------------------+------------------+------------------+------------------+------------------+------------------+------------+------------------
+------------------+--------+-----------------+--------+------------------+-------------------+------------------+--------+------------------+------------------+------------------+------------------+-----------
---------+-----------+
|FL_DATE|OP_CARRIER|OP_CARRIER_FL_NUM|ORIGIN|DEST|CRS_DEP_TIME|          DEP_TIME|         DEP_DELAY|          TAXI_OUT|       WHEELS_OFF|        WHEELS_ON|          TAXI_IN|CRS_ARR_TIME|          ARR_TIME
|         ARR_DELAY|CANCELLED|CANCELLATION_CODE|DIVERTED|    CRS_ELAPSED_TIME|ACTUAL_ELAPSED_TIME|          AIR_TIME|DISTANCE|    CARRIER_DELAY|    WEATHER_DELAY|         NAS_DELAY|   SECURITY_DELAY|LATE_AIRCR
AFT_DELAY|Unnamed: 27|
+-------+----------+-----------------+------+----+------------+------------------+------------------+------------------+------------------+------------------+------------------+------------+------------------
+------------------+--------+-----------------+--------+------------------+-------------------+------------------+--------+------------------+------------------+------------------+------------------+-----------
---------+-----------+
|    0.0|       0.0|              0.0|   0.0| 0.0|         0.0|1.5570505414471807|1.6252149111534209|1.6057512595228411|1.605737396523104|1.653107266624024|1.653107266624024|         0.0|1.653093403624287
|1.8997854839420716|     0.0|98.38379603867556|     0.0|1.386299973688026...| 1.8637694106256566|1.8637694106256566|     0.0|81.2473816259247|81.2473816259247|81.2473816259247|81.2473816259247|   81.2473
816259247|      100.0|
+-------+----------+-----------------+------+----+------------+------------------+------------------+------------------+------------------+------------------+------------------+------------+------------------
+------------------+--------+-----------------+--------+------------------+-------------------+------------------+--------+------------------+------------------+------------------+------------------+-----------
---------+-----------+
```

Appendix 1. Count Number of Null Values

Since we observed that more than 80% of the data was missing for columns **CARRIER_DELAY, WEATHER_DELAY,MS_DELAY,SECURITY_DELAY,Unnamed**, we dropped these columns as they serve no purpose.

Now we tried to Map State and City of each Source and Destination Airports.

```
def map_state_city(self,df1,df2):
    """
    :param df1: Requires DataFrame whose State and City has to be Found
    :param df2: Requires Reference DataFrame of Airport Codes,State,City
    :return: Returns New DataFrame with OriginState, OriginCity, DestinationState, DestiantionCity
    """
    new_df = df1.join(df2,df1.ORIGIN == df2.IATA,"left").drop("Airport","IATA")\
        .withColumnRenamed("State","OriginState").withColumnRenamed("City","OriginCity")\
        .join(df2,df1.DEST == df2.IATA,"left").drop("Airport","IATA")\
        .withColumnRenamed("State","DestinationState").withColumnRenamed("City","DestinationCity")
    new_df.show(5)
    print("\n")
    return new_df
```

Mapping State and City for 2018 year

FL_DATE	OP_CARRIER	OP_CARRIER_FL_NUM	ORIGIN	DEST	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	WHEELS_ON	TAXI_IN	CRS_ARR_TIME	ARR_TIME	ARR_DELAY	CANCELLED	CANCELLATION_CODE	DIVERTED	CRS_ELAPSED_TIME	ACTUAL_ELAPSED_TIME	AIR_TIME	DISTANCE	OriginCity	OriginState	DestinationCity	DestinationState
2018-01-01	UA	2429	EWR	DEN	1517	1512.0	-5.0	15.0	1527.0	1712.0	10.0	1745	1722.0	-23.0	0.0	null	0.0	268.0	250.0	225.0	1605.0	Newark	New Jersey	Denver	Colorado
2018-01-01	UA	2427	LAS	SFO	1115	1107.0	-8.0	11.0	1118.0	1223.0	7.0	1254	1230.0	-24.0	0.0	null	0.0	99.0	83.0	65.0	414.0	Las Vegas	Nevada	San Francisco	California
2018-01-01	UA	2426	SNA	DEN	1335	1330.0	-5.0	15.0	1345.0	1631.0	5.0	1649	1636.0	-13.0	0.0	null	0.0	134.0	126.0	106.0	846.0	Santa Ana	California	Denver	Colorado
2018-01-01	UA	2425	RSW	ORD	1546	1552.0	6.0	19.0	1611.0	1748.0	6.0	1756	1754.0	-2.0	0.0	null	0.0	190.0	182.0	157.0	1120.0	Fort Myers	Florida	Chicago	Illinois
2018-01-01	UA	2424	ORD	ALB	630	650.0	20.0	13.0	703.0	926.0	10.0	922	936.0	14.0	0.0	null	0.0	112.0	106.0	83.0	723.0	Chicago	Illinois	Albany	New York

only showing top 5 rows

Appendix 2. Mapping of each state with cities

Here we can see that each Airport Codes have been mapped to its State and City.

Next for the purpose of finding out if there is relation between Year and whether the flight is delayed or not, we featured engineered our column, i.e we included Year, Month, Date, Day and Day of Week

```
def extract_year_month(self,df):
    """
    :param df: Pass DataFrame whose Year, Month, Day and DayName has to be extarcted
    :return: New DataFrame with Year, Month, Day and DayName
    """
    new_df = df.withColumn("Year",year(df.FL_DATE))\
        .withColumn('Month',month(df.FL_DATE))\
        .withColumn('Day',dayofmonth(df.FL_DATE))\
        .withColumn("Day_Name",date_format(col("FL_DATE"), "EEEE"))\
        .withColumn("Day_OfWeek", dayofweek(df.FL_DATE))
    new_df.show(5)
    print("\n")
    return new_df
```

```
Extracting Year and Month for 2018 year
+----------+----------+-----------------+------+----+------------+--------+---------+--------+----------+----------+--------+------------+--------+---------+---------+-----------------+--------+----------------+-
-----------------+--------+--------+----------+-----------+---------------+----------------+----+-----+---+--------+---------+
|   FL_DATE|OP_CARRIER|OP_CARRIER_FL_NUM|ORIGIN|DEST|CRS_DEP_TIME|DEP_TIME|DEP_DELAY|TAXI_OUT|WHEELS_OFF|WHEELS_ON|TAXI_IN|CRS_ARR_TIME|ARR_TIME|ARR_DELAY|CANCELLED|CANCELLATION_CODE|DIVERTED|CRS_ELAPSED_TIME|A
CTUAL_ELAPSED_TIME|AIR_TIME|DISTANCE|OriginCity|OriginState|DestinationCity|DestinationState|Year|Month|Day|Day_Name|Day_OfWeek|
+----------+----------+-----------------+------+----+------------+--------+---------+--------+----------+----------+--------+------------+--------+---------+---------+-----------------+--------+----------------+-
-----------------+--------+--------+----------+-----------+---------------+----------------+----+-----+---+--------+---------+
|2018-01-01|        UA|             2429|   EWR| DEN|        1517|  1512.0|     -5.0|    15.0|    1527.0|   1712.0|   10.0|        1745|  1722.0|    -23.0|      0.0|             null|     0.0|           268.0|
            250.0|   225.0|  1605.0|    Newark| New Jersey|         Denver|        Colorado|2018|    1|  1|  Monday|         2|
|2018-01-01|        UA|             2427|   LAS| SFO|        1115|  1107.0|     -8.0|    11.0|    1118.0|   1223.0|    7.0|        1254|  1230.0|    -24.0|      0.0|             null|     0.0|            99.0|
             83.0|    65.0|   414.0| Las Vegas|     Nevada|  San Francisco|      California|2018|    1|  1|  Monday|         2|
|2018-01-01|        UA|             2426|   SNA| DEN|        1335|  1330.0|     -5.0|    15.0|    1345.0|   1631.0|    5.0|        1649|  1636.0|    -13.0|      0.0|             null|     0.0|           134.0|
            126.0|   106.0|   846.0| Santa Ana| California|         Denver|        Colorado|2018|    1|  1|  Monday|         2|
|2018-01-01|        UA|             2425|   RSW| ORD|        1546|  1552.0|      6.0|    19.0|    1611.0|   1748.0|    6.0|        1756|  1754.0|     -2.0|      0.0|             null|     0.0|           190.0|
            182.0|   157.0|  1120.0|Fort Myers|    Florida|        Chicago|        Illinois|2018|    1|  1|  Monday|         2|
|2018-01-01|        UA|             2424|   ORD| ALB|         630|   650.0|     20.0|    13.0|     703.0|    926.0|   10.0|         922|   936.0|     14.0|      0.0|             null|     0.0|           112.0|
            106.0|    83.0|   723.0|   Chicago|   Illinois|         Albany|        New York|2018|    1|  1|  Monday|         2|
+----------+----------+-----------------+------+----+------------+--------+---------+--------+----------+----------+--------+------------+--------+---------+---------+-----------------+--------+----------------+-
-----------------+--------+--------+----------+-----------+---------------+----------------+----+-----+---+--------+---------+
only showing top 5 rows
```

Appendix 3. Extracting Year and Month

Now we wanted to know, whether Season has effects on Flight being late or not. Hence we feature engineered new column where we stored Seasons information.

```python
def seasons(self,sc,df):
    df.createOrReplaceTempView("temp_df")
    new_df = sc.sql("SELECT *, \
                          CASE \
                          WHEN Month IN (3,4,5)   THEN  'SPRING'  \
                          WHEN Month IN (6,7,8)   THEN  'SUMMER'  \
                          WHEN Month IN (9,10,11) THEN  'AUTUMN'  \
                          WHEN Month IN (12,1,2)  THEN  'WINTER'  \
                          END AS Seasons \
                      FROM temp_df")
    new_df.show(5)
    print("\n")
    return new_df
```

```
Extarcting Seasons for 2018 year
+----------+----------+-----------------+------+----+------------+--------+---------+--------+----------+---------+-------+------------+--------+---------+---------+-----------------+--------+----------------+-
-----------------+--------+--------+----------+-----------+---------------+----------------+----+-----+---+--------+----------+-------+
|   FL_DATE|OP_CARRIER|OP_CARRIER_FL_NUM|ORIGIN|DEST|CRS_DEP_TIME|DEP_TIME|DEP_DELAY|TAXI_OUT|WHEELS_OFF|WHEELS_ON|TAXI_IN|CRS_ARR_TIME|ARR_TIME|ARR_DELAY|CANCELLED|CANCELLATION_CODE|DIVERTED|CRS_ELAPSED_TIME|A
CTUAL_ELAPSED_TIME|AIR_TIME|DISTANCE|OriginCity|OriginState|DestinationCity|DestinationState|Year|Month|Day|Day_Name|Day_OfWeek|Seasons|
+----------+----------+-----------------+------+----+------------+--------+---------+--------+----------+---------+-------+------------+--------+---------+---------+-----------------+--------+----------------+-
-----------------+--------+--------+----------+-----------+---------------+----------------+----+-----+---+--------+----------+-------+
|2018-01-01|        UA|             2429|   EWR| DEN|        1517|  1512.0|     -5.0|    15.0|    1527.0|   1712.0|   10.0|        1745|  1722.0|    -23.0|      0.0|             null|     0.0|           268.0|
            250.0|   225.0|  1605.0|    Newark| New Jersey|         Denver|        Colorado|2018|    1|  1|  Monday|         2| WINTER|
|2018-01-01|        UA|             2427|   LAS| SFO|        1115|  1107.0|     -8.0|    11.0|    1118.0|   1223.0|    7.0|        1254|  1230.0|    -24.0|      0.0|             null|     0.0|            99.0|
             83.0|    65.0|   414.0| Las Vegas|     Nevada|  San Francisco|      California|2018|    1|  1|  Monday|         2| WINTER|
|2018-01-01|        UA|             2426|   SNA| DEN|        1335|  1330.0|     -5.0|    15.0|    1345.0|   1631.0|    5.0|        1649|  1636.0|    -13.0|      0.0|             null|     0.0|           134.0|
            126.0|   106.0|   846.0| Santa Ana| California|         Denver|        Colorado|2018|    1|  1|  Monday|         2| WINTER|
|2018-01-01|        UA|             2425|   RSW| ORD|        1546|  1552.0|      6.0|    19.0|    1611.0|   1748.0|    6.0|        1756|  1754.0|     -2.0|      0.0|             null|     0.0|           190.0|
            182.0|   157.0|  1120.0|Fort Myers|    Florida|        Chicago|        Illinois|2018|    1|  1|  Monday|         2| WINTER|
|2018-01-01|        UA|             2424|   ORD| ALB|         630|   650.0|     20.0|    13.0|     703.0|    926.0|   10.0|         922|   936.0|     14.0|      0.0|             null|     0.0|           112.0|
            106.0|    83.0|   723.0|   Chicago|   Illinois|         Albany|        New York|2018|    1|  1|  Monday|         2| WINTER|
+----------+----------+-----------------+------+----+------------+--------+---------+--------+----------+---------+-------+------------+--------+---------+---------+-----------------+--------+----------------+-
-----------------+--------+--------+----------+-----------+---------------+----------------+----+-----+---+--------+----------+-------+
only showing top 5 rows
```

Appendix 4. Adding Seasons column

Now we wanted to know, whether flight being delayed is in Morning or Evening, Is there a particular time of the day, when more flights are delayed

```python
def parts_of_day(self,sc,df):
    df.createOrReplaceTempView("temp_df")
    new_df = sc.sql("SELECT *, \
                        CASE \
                        WHEN CRS_DEP_TIME >= 0000 AND CRS_DEP_TIME < 400   THEN  'Early Morning'  \
                        WHEN CRS_DEP_TIME >= 400 AND CRS_DEP_TIME < 1200   THEN  'Morning'  \
                        WHEN CRS_DEP_TIME >= 1200 AND CRS_DEP_TIME < 1800   THEN  'Afternoon'  \
                        WHEN CRS_DEP_TIME >= 1800 AND CRS_DEP_TIME <= 2359   THEN  'Evening/Night'  \
                        END AS Parts_of_day \
                    FROM temp_df")
    new_df.show(5)
    print("\n")
    return new_df
```

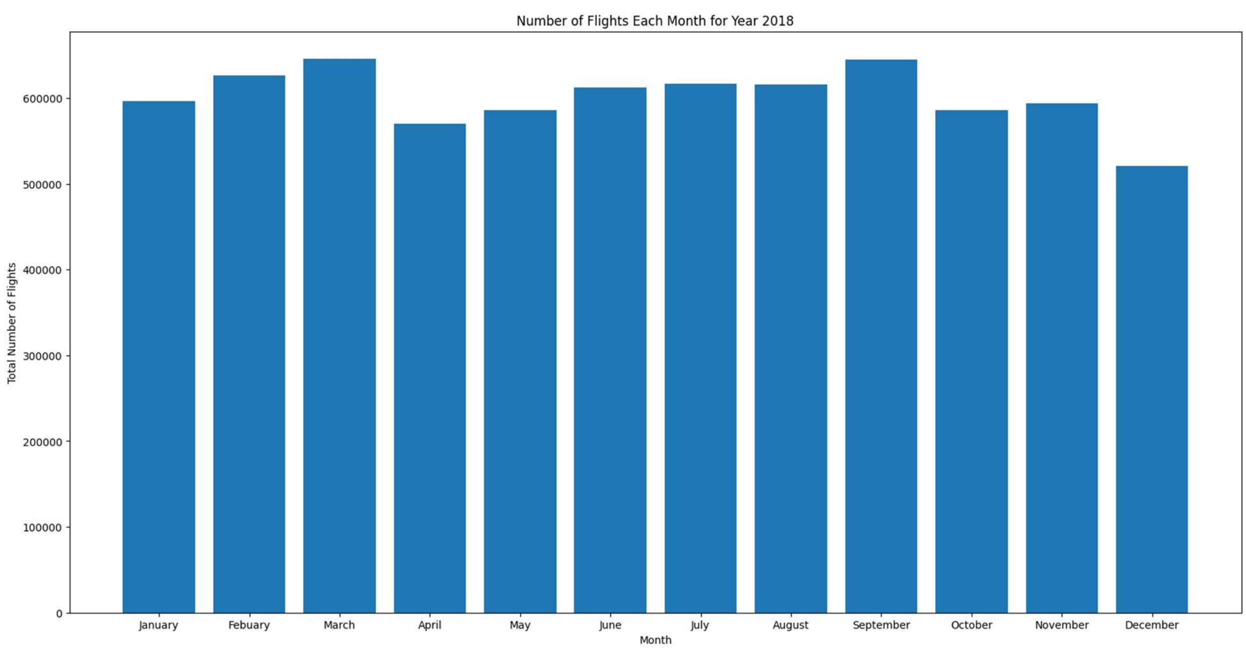
Extarcting Parts of Day for 2018 year

FL_DATE	OP_CARRIER	OP_CARRIER_FL_NUM	ORIGIN	DEST	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	WHEELS_ON	TAXI_IN	CRS_ARR_TIME	ARR_TIME	ARR_DELAY	CANCELLED	CANCELLATION_CODE	DIVERTED	CRS_ELAPSED_TIME	ACTUAL_ELAPSED_TIME	AIR_TIME	DISTANCE	OriginCity	OriginState	DestinationCity	DestinationState	Year	Month	Day	Day_Name	Day_OfWeek	Seasons	Parts_of_day
2018-01-01	UA	2429	EWR	DEN	1517	1512.0	-5.0	15.0	1527.0	1712.0	10.0	1745	1722.0	-23.0	0.0	null	0.0	268.0	250.0	225.0	1605.0	Newark	New Jersey	Denver	Colorado	2018	1	1	Monday	2	WINTER	Afternoon
2018-01-01	UA	2427	LAS	SFO	1115	1107.0	-8.0	11.0	1118.0	1223.0	7.0	1254	1230.0	-24.0	0.0	null	0.0	99.0	83.0	65.0	414.0	Las Vegas	Nevada	San Francisco	California	2018	1	1	Monday	2	WINTER	Morning
2018-01-01	UA	2426	SNA	DEN	1335	1330.0	-5.0	15.0	1345.0	1631.0	5.0	1649	1636.0	-13.0	0.0	null	0.0	134.0	126.0	106.0	846.0	Santa Ana	California	Denver	Colorado	2018	1	1	Monday	2	WINTER	Afternoon
2018-01-01	UA	2425	RSW	ORD	1546	1552.0	6.0	19.0	1611.0	1748.0	6.0	1756	1754.0	-2.0	0.0	null	0.0	190.0	182.0	157.0	1120.0	Fort Myers	Florida	Chicago	Illinois	2018	1	1	Monday	2	WINTER	Afternoon
2018-01-01	UA	2424	ORD	ALB	630	650.0	20.0	13.0	703.0	926.0	10.0	922	936.0	14.0	0.0	null	0.0	112.0	106.0	83.0	723.0	Chicago	Illinois	Albany	New York	2018	1	1	Monday	2	WINTER	Morning

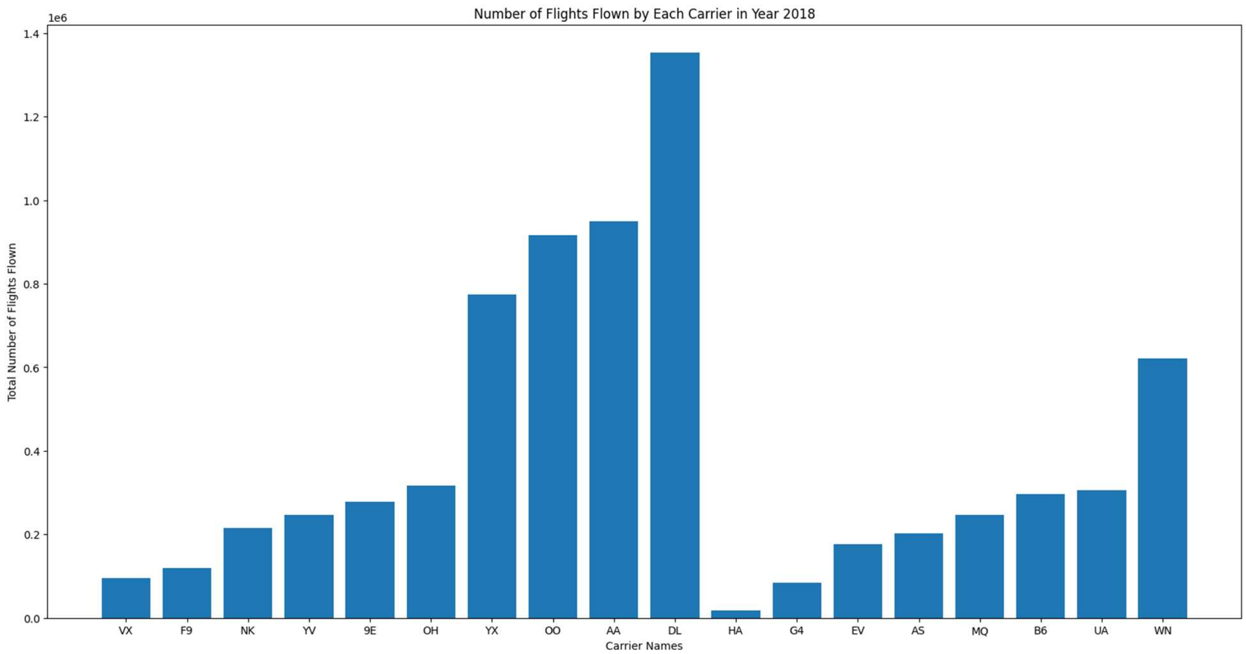
only showing top 5 rows

Appendix 5. Categorized delay according to session of day

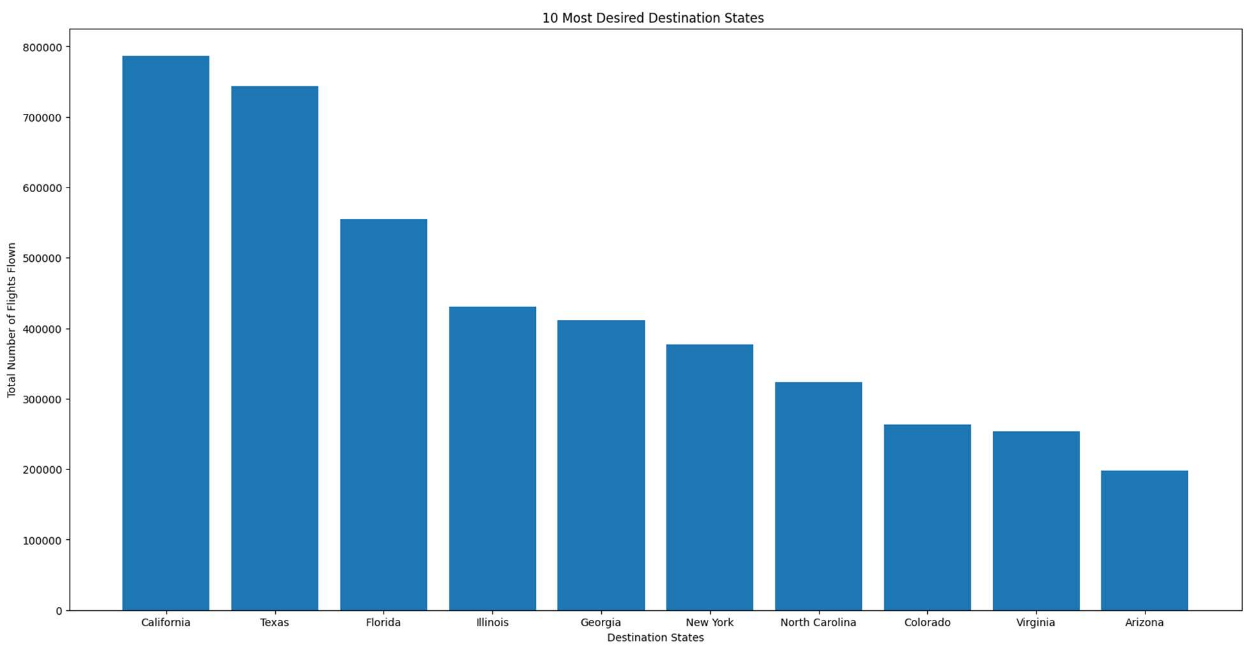
Data Visualizations



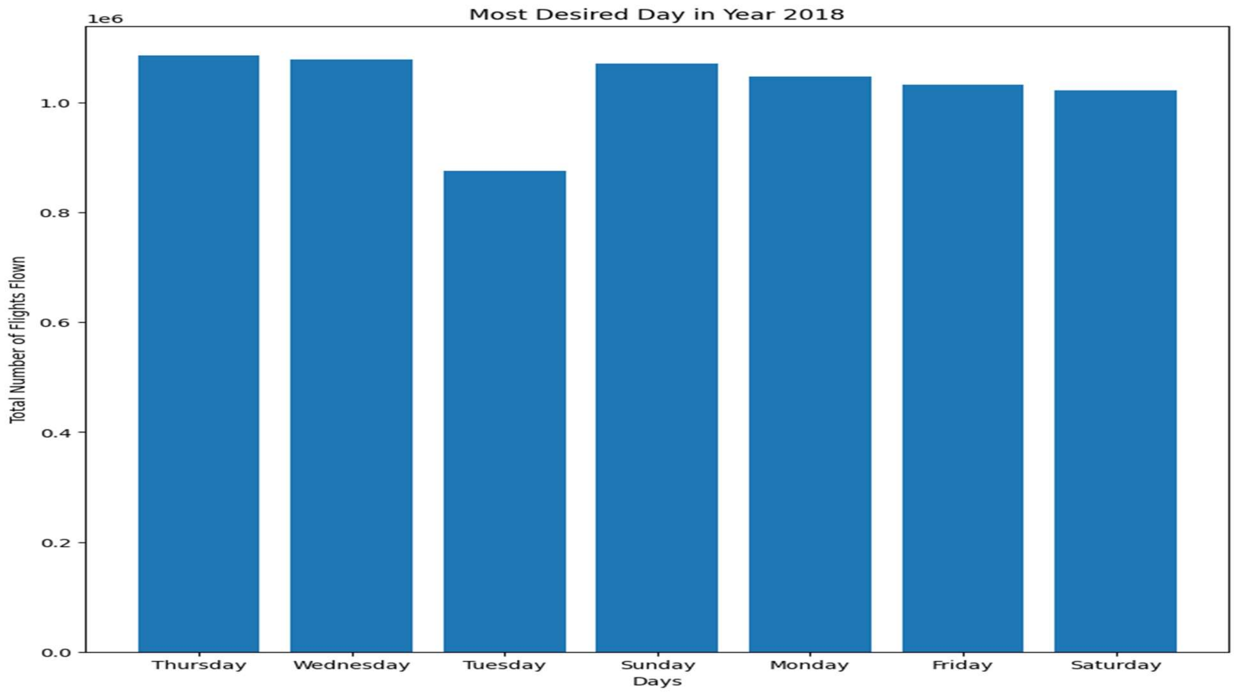
Appendix 6. Bar chart for No. of flights operated for each month for year 2018



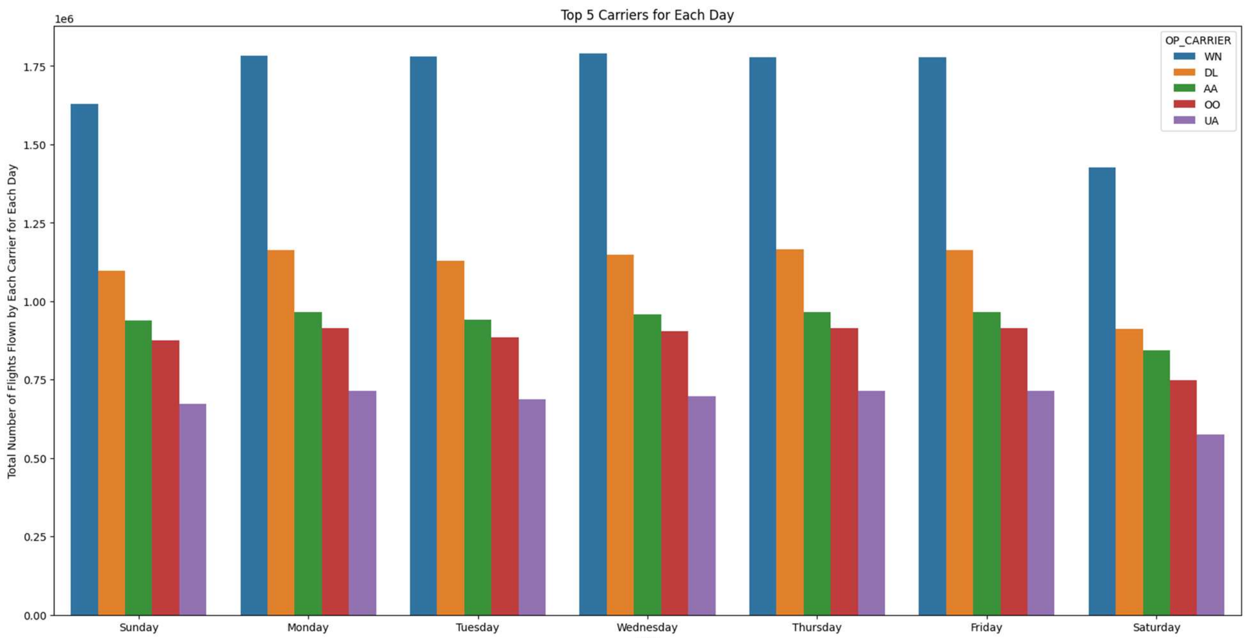
Appendix 7. Bar chart for No. of flights flown per carrier in year 2018



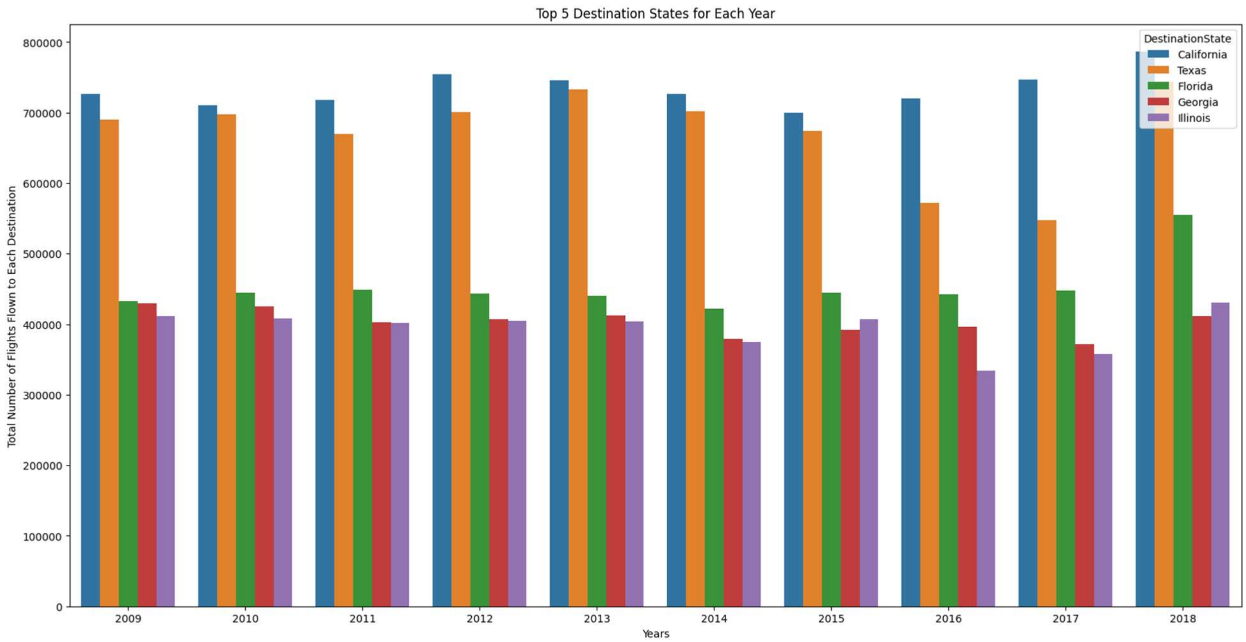
Appendix 8. Bar chart for Most Desired Destinations



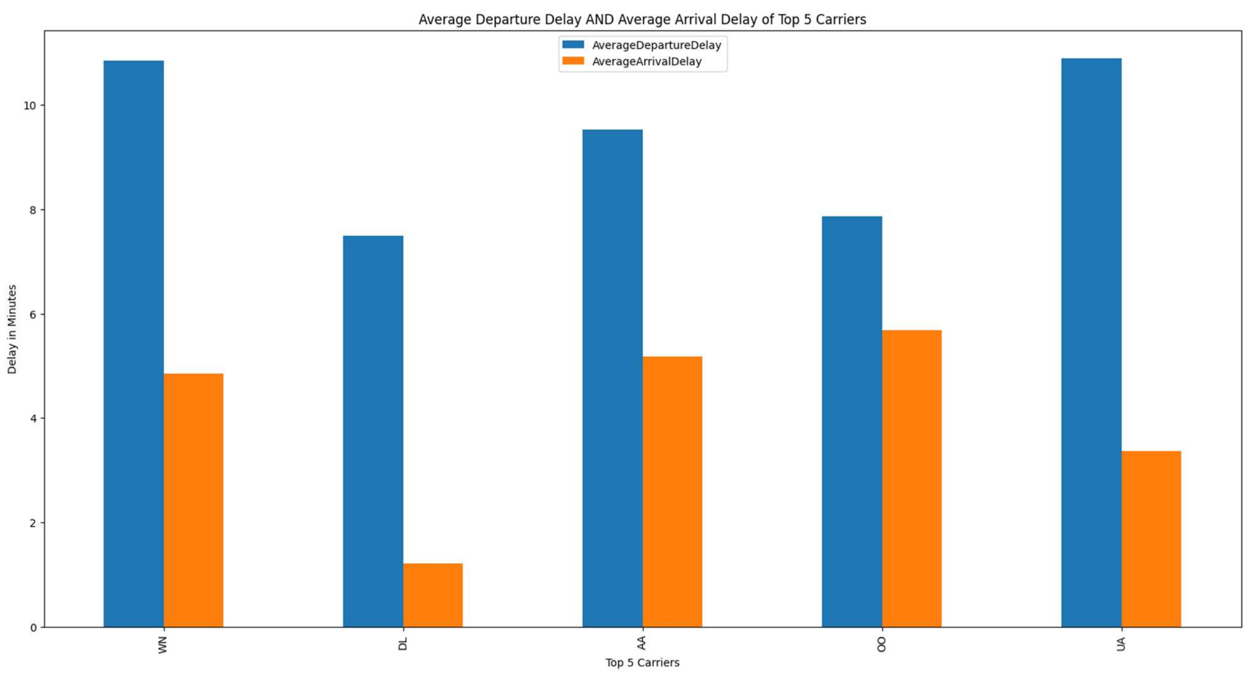
Appendix 9. Bar chart for most No. of flights flown on a Day in year 2018



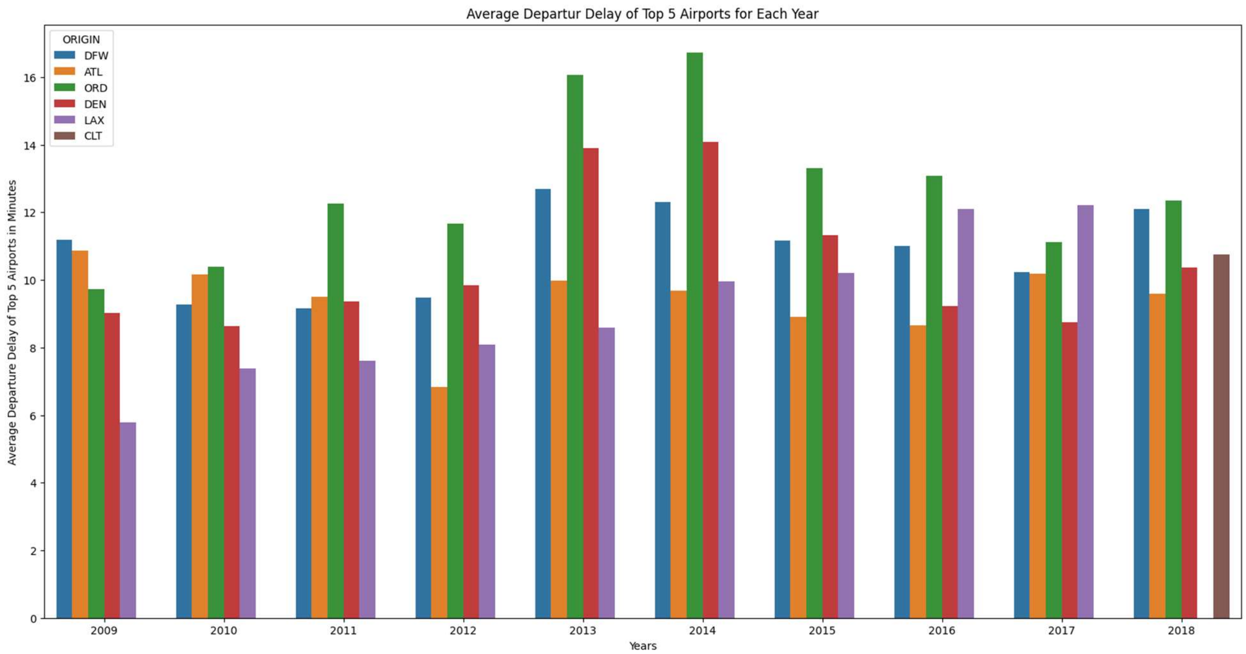
Appendix 10. Bar chart for Top 5 Carriers for each day



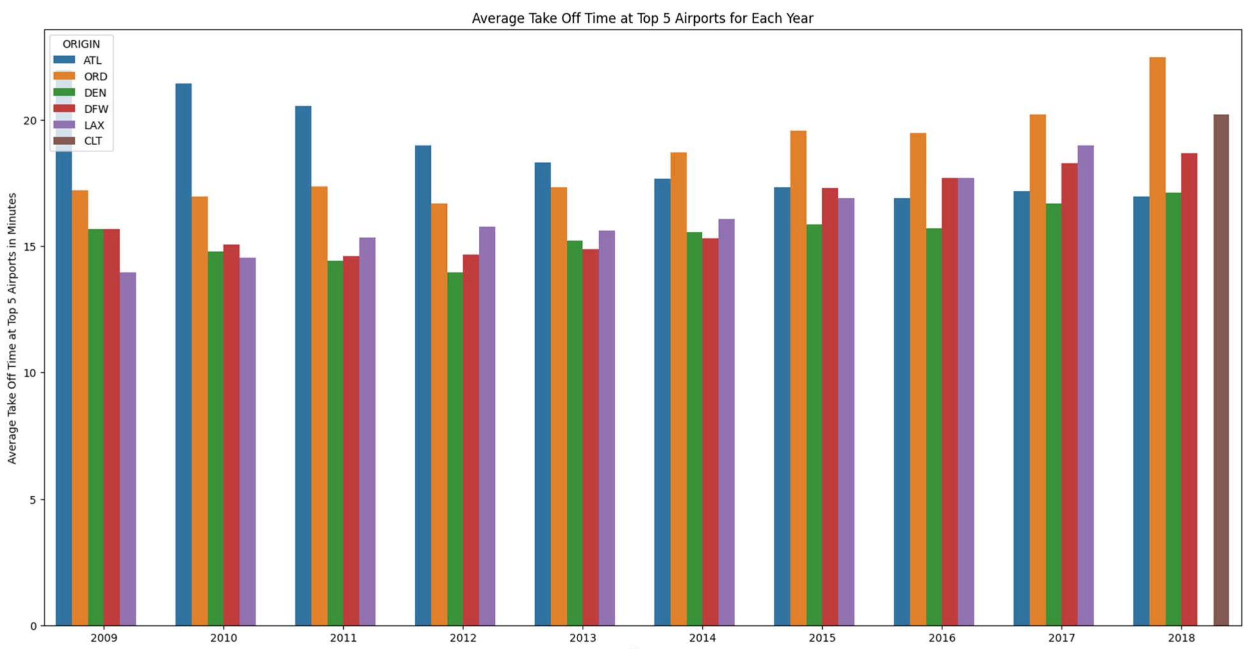
Appendix 11. Bar chart for Top 5 destinations states each year



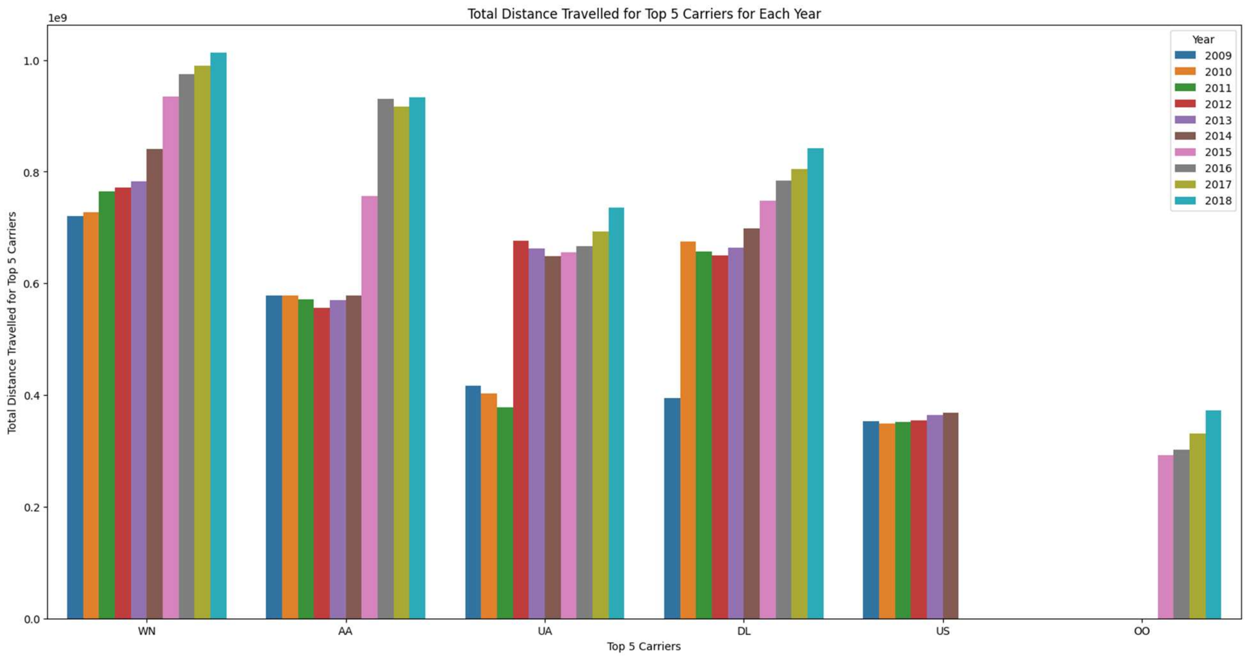
Appendix 12. Bar chart for Average Arrival Delay and Average Departure Delay of Top 5 Carriers



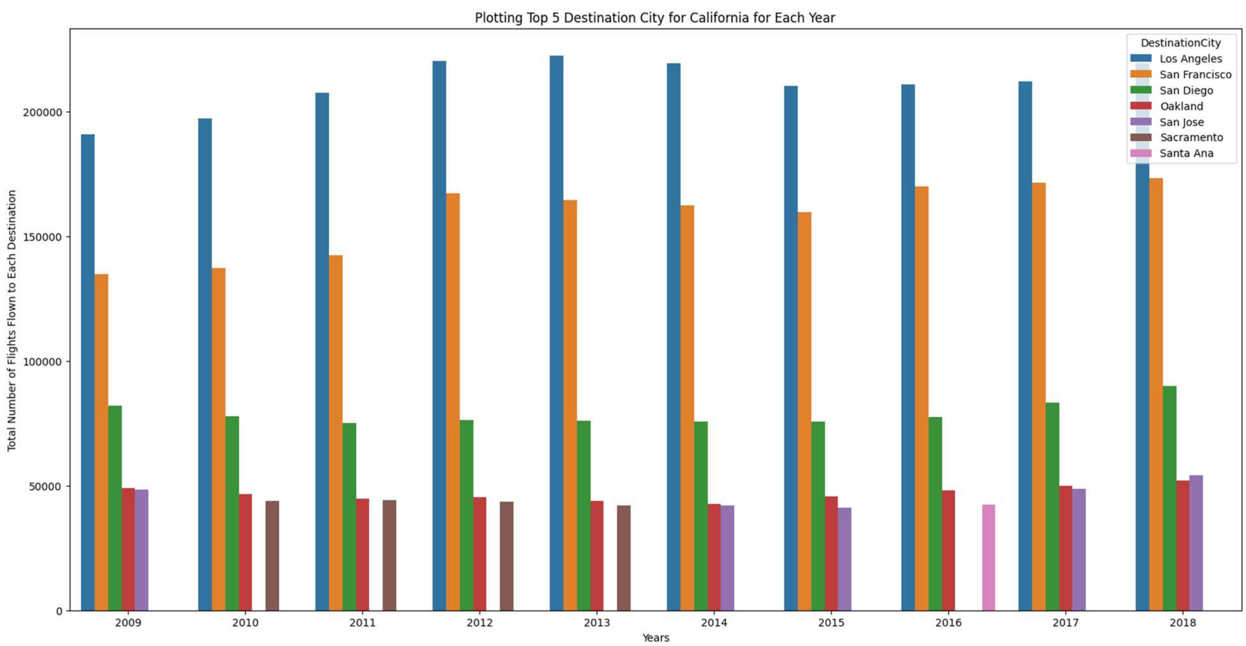
Appendix 13. Bar chart for Average Departure Delay of Top 5 Airport



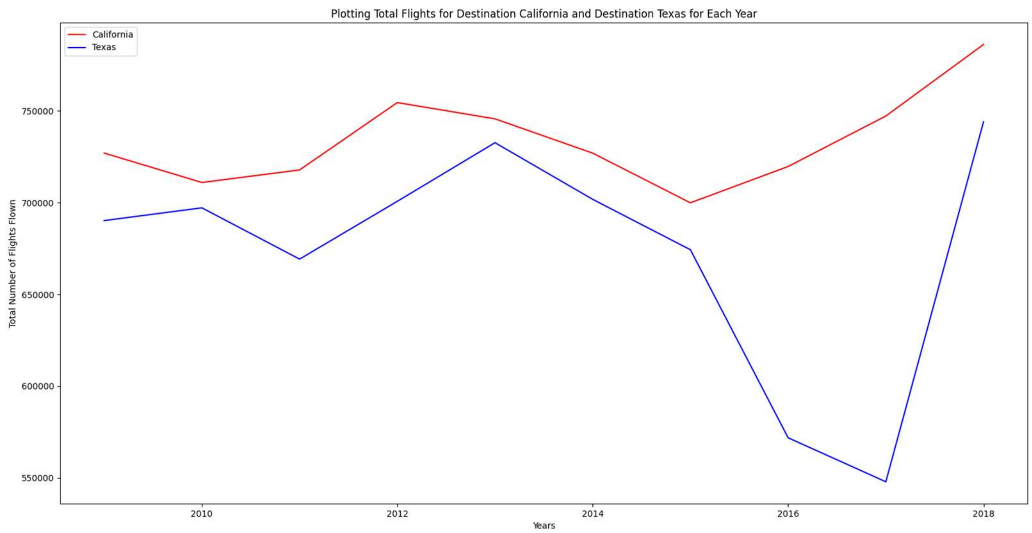
Appendix 14. Bar chart for Average Take Off Time of Top 5 Airports



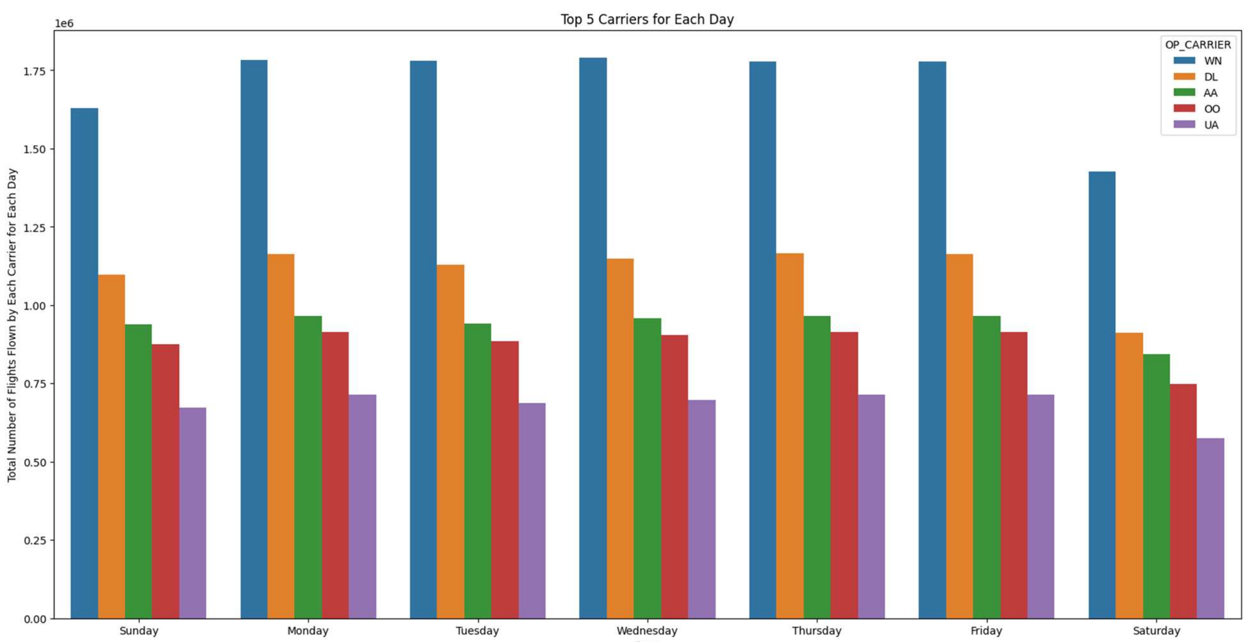
Appendix 15. Bar chart for Total Distance Traveled for Top 5 Carriers



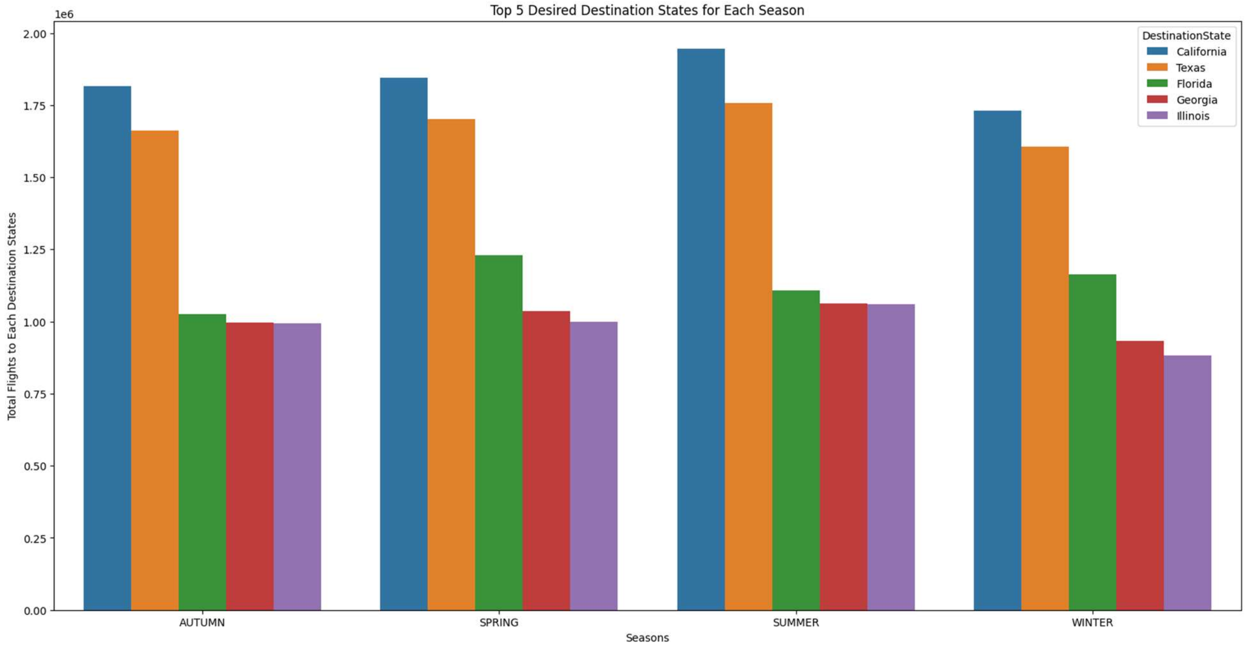
Appendix 16. Top 5 Destination City for California Each Year



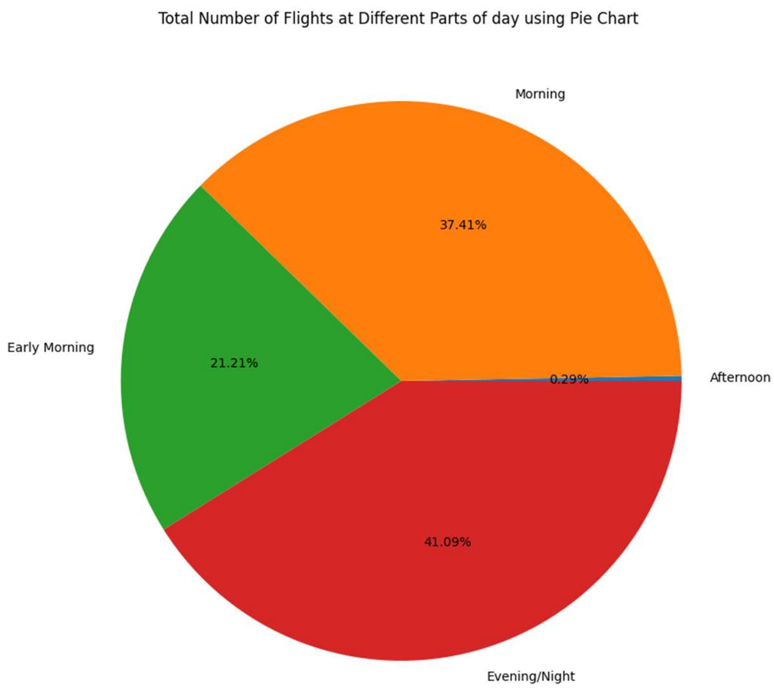
Appendix 17. Line chart for Destination California and Destination Texas



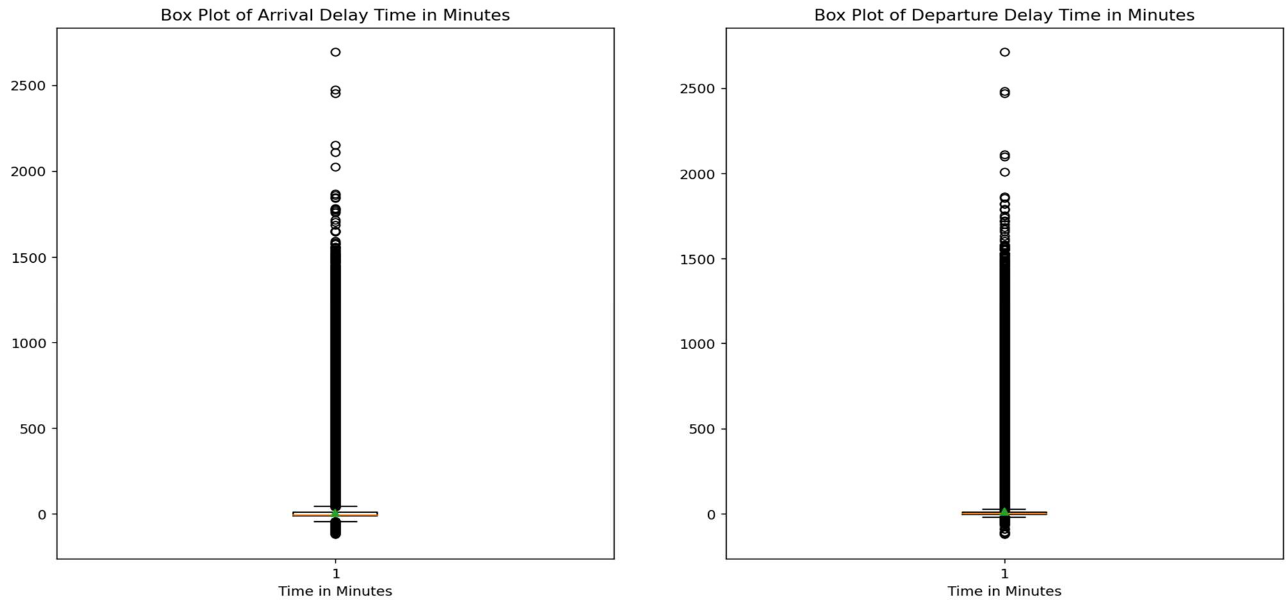
Appendix 18. Top 5 Carriers for Each Day



Appendix 19. Top 5 Desired Destination States Each Season

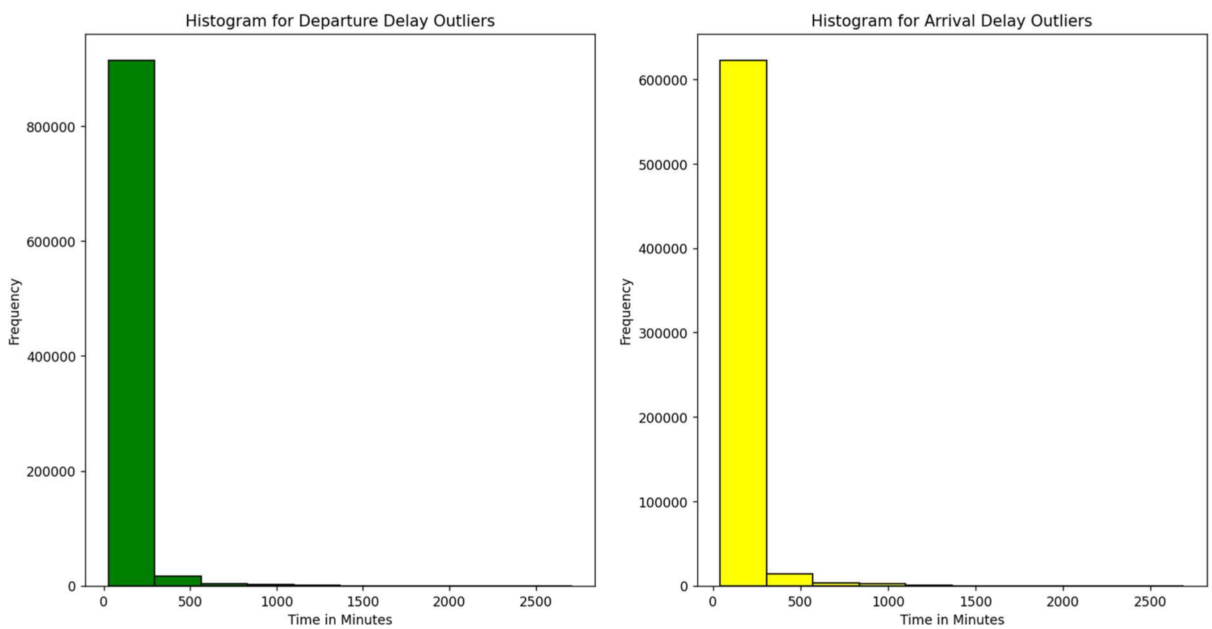


Appendix 20. Pie chart Total No. of Flights at Different Parts of Day



Appendix 21. Box Plot for Outliers

Since box plot takes account into interquartile range and our dataset contains values positive as well as negative data points and that is why we have very minimal iqr range which results in showing higher number of values as outliers. We have chosen 400 minutes, which is nearly 7 hours as the maximum number. Also we can see from the plot that maximum numbers are concentrated near 0 and 500 that is why we have chosen 400.



Appendix 22. Histogram for Outliers

Above histogram shows the count of every point in the departure delay and arrival delay only for positive values as negative values had the highest number as -100.

Since we are trying to classify into 3 Class, We have chosen if Arrival Delay time is less than -10 as Early and Arrival Delay time greater than 10 as Late.

```
def create_labels(self,df):
    new_df = df.withColumn("Labels", when(df["ARR_DELAY"] < -10, lit("Early")).when(df["ARR_DELAY"] > 10,lit("Delay"))\
                           .otherwise(lit("OnTime")))

    return new_df
```

Inorder to prepare our model we selected these features:-
OP_CARRIER, TAXI_OUT, WHEELS_OFF, WHEELS_ON, TAXI_IN, DEP_TIME, DEP_DELAY, Labels.

Model Preparation

Inorder to prepare our Model, we created a pipeline. This Pipeline includes StringIndexer, OneHotEncoder, VectorAssembler and then inserting this into a Pipeline.

```
# Creating a String Indexer for Features Column
self.stringIndexer = StringIndexer(inputCols=category_columns, outputCols=indexoutputcols, handleInvalid='skip')
print("String Indexing Done")

# Creating String Indexer for Labels Column
self.labels_stringindexer = StringIndexer(inputCol='Labels', outputCol='label')
print("Label String Indexing  Done")

# Creating One Hot Encoder
self.oheEncoder = OneHotEncoder(inputCols=indexoutputcols, outputCols=oheoutputcols)
print("One Hot Encoding Done")

# Creating Vector Assembler
assemblerinputs = oheoutputcols + numeric_col
self.vectorassembler = VectorAssembler(inputCols=assemblerinputs, outputCol='features',handleInvalid='skip')
print("Vector Assembling Done")

self.stages = [self.stringIndexer, self.labels_stringindexer, self.oheEncoder, self.vectorassembler]
print("Created a Stages Pipeline")

stagespipeline = Pipeline(stages=self.stages)
stagespipelinemodel = stagespipeline.fit(self.df)
self.temp_df = stagespipelinemodel.transform(self.df)
final_cols = cols + ['features','label']
self.new_df = self.temp_df.select(final_cols)
print("Stages PipeLine Transformed")
```

Decision Tree Algorithm:

```python
def decision_tree_classifier(self):
    start_time = time.time()
    #Creating Decision Tree Algorithm
    print("Using Decision Tree Classifier Algorithm")
    dt = DecisionTreeClassifier(labelCol='label',featuresCol='features',maxDepth=3)
    DT_Model = dt.fit(self.train)
    pred = DT_Model.transform(self.test)

    #Calculate Accuracy
    acc = self.classification_accuracy(pred)

    dt_elapsed = time.strftime("%H:%M:%S", time.gmtime(time.time() - start_time))
    print(f"Decision Tree Algorithm Took: {dt_elapsed}")


    return acc,dt_elapsed
```

```
Accuracy =  0.6732897584591979
Decision Tree Algorithm Took: 00:05:20
```

Random Forest Algorithm:

```python
def random_forest_classifier(self):
    start_time = time.time()
    #Creating Random Forest Algorithm
    print("Using Random Forest Algorithm")
    rf = RandomForestClassifier(labelCol='label',featuresCol='features')
    RF_Model = rf.fit(self.train)
    pred = RF_Model.transform(self.test)

    # Calculate Accuracy
    acc = self.classification_accuracy(pred)


    rf_elapsed = time.strftime("%H:%M:%S", time.gmtime(time.time() - start_time))
    print(f"Random Forest Algorithm Took: {rf_elapsed}")

    return acc,rf_elapsed
```

```
Accuracy =  0.42736962005167484
Random Forest Algorithm Took: 00:06:38
```

Logistic Regression Algorithm:

```python
def logistic_regression(self):
    start_time = time.time()
    print("Using Logistic Regression Algorithm")
    lr = LogisticRegression(featuresCol='features', labelCol='label', maxIter=10)
    ovr = OneVsRest(classifier=lr)
    ovrModel = ovr.fit(self.train)
    pred = ovrModel.transform(self.test)

    acc = self.classification_accuracy(pred)

    lr_elapsed = time.strftime("%H:%M:%S", time.gmtime(time.time() - start_time))
    print(f"Logistic Reression Took: {lr_elapsed}")

    return acc,lr_elapsed
```

```
Accuracy =  0.7302626020216336
Logistic Reression Took: 00:04:10
```

SVM Algorithm

```python
def SVM(self):
    start_time = time.time()
    print("Using SVM Algorithm")
    lsvc = LinearSVC(featuresCol='features', labelCol='label',maxIter=10, regParam=0.1)
    ovr = OneVsRest(classifier=lsvc)
    ovrModel = ovr.fit(self.train)
    pred = ovrModel.transform(self.test)

    acc = self.classification_accuracy(pred)

    svm_elapsed = time.strftime("%H:%M:%S", time.gmtime(time.time() - start_time))
    print(f"SVM Took: {svm_elapsed}")

    return acc, svm_elapsed
```

```
Accuracy =  0.6354342949013073
SVM Took: 00:04:00
```

Comparative Results Machine Learning Algorithms with different No. of Instances:

emr 6.4.0 m4.xlarge 3 instances (1 Master, 2 Core):

```
                         Algorithm  Accuracy Time Spent
0         Decision Tree Algorithm  0.672869   00:07:39
1         Random Forest Algorithm  0.369466   00:09:53
2   Logistic Regression Algorithm  0.731208   00:06:02
3                  SVM Classifier  0.635775   00:05:46
Entire Code Took: 00:33:30
Press Enter to end SparkSession:
```

emr 6.4.0 m4.xlarge 4 instances (1 Master, 3 Core):

```
                         Algorithm  Accuracy Time Spent
0         Decision Tree Algorithm  0.673213   00:05:18
1         Random Forest Algorithm  0.429769   00:06:50
2   Logistic Regression Algorithm  0.731096   00:04:04
3                  SVM Classifier  0.635194   00:04:00
Entire Code Took: 00:23:46
```

emr 6.4.0 m4.xlarge 5 instances (1 master, 4 Core):

```
                         Algorithm  Accuracy Time Spent
0         Decision Tree Algorithm  0.673290   00:05:20
1         Random Forest Algorithm  0.427370   00:06:38
2   Logistic Regression Algorithm  0.730263   00:04:10
3                  SVM Classifier  0.635434   00:04:00
Entire Code Took: 00:23:13
```

Conclusion

From the above comparison we can see that with 5 instances, the time required by each algorithm is almost similar with 4 instances which is less than 3 instance configuration.

Logistic regression (one vs rest) performs better than the other three algorithms. Although we can perform the hyperparameter tuning for all of these algorithms to find the best parameters to get better accuracy. But with these number of instances it was taking a higher time to run the grid search and for the experiments.

References

[1]https://developer.hpe.com/blog/spark-101-what-is-it-what-it-does-and-why-it-matters/
[2]https://spark.apache.org/
[3]https://en.wikipedia.org/wiki/Apache_Spark
[4]https://www.transtats.bts.gov/Homepage.asp
[5]https://iopscience.iop.org/article/10.1088/1755-1315/81/1/012198
[6]https://www.hindawi.com/journals/jat/2021/4292778/

Appendix

1. Count Number of Null Values
2. Mapping of each state with cities
3. Extracting Year and Month
4. Adding Seasons column
5. Categorized delay according to session of day
6. Bar chart for No. of flights operated for each month for year 2018
7. Bar chart for No. of flights flown per carrier in year 2018
8. Bar chart for Most Desired Destinations
9. Bar chart for most No. of flights flown on a Day in year 2018
10. Bar chart for Top 5 Carriers for each day
11. Bar chart for Top 5 destinations states each year
12. Bar chart for Average Arrival Delay and Average Departure Delay of Top 5 Carriers
13. Bar chart for Average Departure Delay of Top 5 Airport
14. Bar chart for Average Take Off Time of Top 5 Airports
15. Bar chart for Total Distance Traveled for Top 5 Carriers
16. Top 5 Destination City for California Each Year
17. Line chart for Destination California and Destination Texas
18. Top 5 Carriers for Each Day
19. Top 5 Desired Destination States Each Season
20. Pie chart Total No. of Flights at Different Parts of Day
21. Box Plot for Outliers
22. Histogram for Outliers