1. # In[1]:
3. #Import the required packages
5. # Data manipulation
6. **import** pandas as pd
7. **import** numpy as np
9. # Filehandling
10. **import** os
12. # Plotting
13. **import** matplotlib.pyplot as plt
14. plt.rcParams['figure.figsize'] = [8.0, 5.0]
16. # Statsmodels libraries
17. **import** statsmodels.api as sm
18. **from** statsmodels.tsa.stattools **import** pacf
19. **from** statsmodels.graphics.tsaplots **import** plot\_pacf
20. **from** statsmodels.tsa.seasonal **import** seasonal\_decompose
21. **from** statsmodels.tsa.stattools **import** acf
22. **from** statsmodels.graphics.tsaplots **import** plot\_acf
23. **from** statsmodels.tsa.arima\_model **import** ARIMA
24. **from** statsmodels.tsa.stattools **import** adfuller

27. # In[2]:
29. firstfile = 'C:/Users/durandt/Documents/Aston/Data Sets/On-Time Data/csv/On\_Time\_On\_Time\_Performance\_2017\_1.csv'
30. df1 = pd.read\_csv(firstfile, low\_memory=False)
31. df1.head()

34. # In[3]:
36. # NULLs
37. info=pd.DataFrame(df1.isnull().sum()).T.rename(index={0:'# NULL'})
38. info=info.append(pd.DataFrame(df1.isnull().sum()/df1.shape[0]\*100).T.rename(index={0:'% NULL'}))
39. info=info.append(pd.DataFrame(df1.dtypes).T.rename(index={0:'type'}))
40. AllNull = info.T
41. AllNull = AllNull[AllNull['% NULL'] == 100]
42. **print**("Columns with all nulls:")
43. list(AllNull.index)

46. # In[4]:
48. # Subset the data
49. df1 = df1[['Year','Quarter','Month','DayofMonth','DayOfWeek','FlightDate','UniqueCarrier','Origin','Dest','CRSDepTime','DepTime'
50. ,'DepDelay','TaxiOut','WheelsOff','Cancelled','Diverted']]

53. # In[5]:
55. # Show the count of flights per Operator
56. df1['UniqueCarrier'].value\_counts().plot(kind='bar')
57. plt.show()

60. # In[6]:
62. **print**('Top 20 Origins')
63. df1.groupby('Origin').agg({'Origin':'count'}).sort\_values('Origin',
64. ascending = False).rename(columns = {'Origin' : 'Count'}).head(20).transpose()

67. # In[7]:
69. **print**('Top 10 Origins for American Airlines')
70. df1[df1['UniqueCarrier']=='AA'].groupby('Origin').agg({'Origin':'count'}).sort\_values('Origin',
71. ascending = False).rename(columns = {'Origin' : 'Count'}).head(10).transpose()

74. # In[8]:
76. df1 = df1[(df1.UniqueCarrier =='AA') & (df1.Origin =='JFK') & (df1.Cancelled == 0.0)]

79. # In[9]:
81. **print**("For example - Number of Flights on 01/01/2017: "
82. + df1[(df1.FlightDate =='2017-01-01')]['DepTime'].count().astype(str))

85. # In[10]:
87. # Create a function to retain only the hour portion of the timeband
88. **def** converttimeband(time):
89. **if** pd.isnull(time):
90. **return** np.nan
91. **elif** len(time) > 5:
92. timeband = time[0:2]
93. **elif** len(time) > 4:
94. timeband = "0" + time[0:1]
95. **else**:
96. timeband = '00'
97. **return**(timeband)
99. # Create the Timeband column from Departure Time converted to a string
100. df1['Timeband'] = df1['DepTime'].astype(str)
102. # Apply the function to the new Timeband column
103. df1['Timeband'] = df1['Timeband'].apply(converttimeband)

106. # In[11]:
108. # Show the count of flights departing in each Timeband
109. df1[(df1.FlightDate =='2017-01-01')].groupby('Timeband').agg({'Timeband':'count'}).plot(kind='bar',sort\_columns = 'Timeband')
110. plt.ylabel("Flight Count")
111. plt.legend(loc='best')
112. plt.show()

115. # In[12]:
117. # Show the mean of flights departing in each Timeband
118. df1[(df1.FlightDate =='2017-01-01')].groupby('Timeband').agg({'TaxiOut':'mean'}).plot(kind='bar',sort\_columns = 'Timeband')
119. plt.ylabel("Taxi Out Time Mean")
120. plt.legend(loc='best')
121. plt.show()

124. # In[13]:
126. # Combine all files with the cutdown list of columns
127. datadir = 'C:/Users/durandt/Documents/Aston/Data Sets/On-Time Data/csv/'
128. filelist = os.listdir(datadir)
129. dfpart = [pd.read\_csv(datadir + file, usecols = ['Year','Quarter','Month','DayofMonth','UniqueCarrier','Origin','DepTime','TaxiOut','Cancelled']) **for** file **in** filelist]
130. df = pd.concat(dfpart,ignore\_index=True)
132. # Filter the data to AA and JFK
133. df = df[(df.UniqueCarrier =='AA') & (df.Origin =='JFK') & (df.Cancelled == 0.0)]
135. # Create a new FlightDate from the 'Year','Month','DayofMonth' as there are different formats of FlightDate across source files
136. df['FlightDate'] = df['Year'].astype(str) + "-" + df['Month'].astype(str) + "-" + df['DayofMonth'].astype(str)
138. # Create the Timeband column from Departure Time converted to a string
139. df['Timeband'] = df['DepTime'].astype(str)
141. # Apply the function to the new Timeband column
142. df['Timeband'] = df['Timeband'].apply(converttimeband)
144. # Remove the category columns that are no longer relavent
145. df = df[['FlightDate','DepTime','TaxiOut','Timeband']]

148. # In[14]:
150. # Create the Time from DepTime converted to a string
151. df['Time'] = df['DepTime'].astype(str)
153. # Create a function to convert to time
154. **def** converttime(time):
155. **if** pd.isnull(time):
156. **return** "00:00"
157. **if** time == "2400.0":
158. **return** "00:00"
159. **elif** len(time) > 5:
160. hrpart = time[0:2]
161. minpart = time[2:4]
162. **elif** len(time) > 4:
163. hrpart = "0" + time[0:1]
164. minpart = time[1:3]
165. **elif** len(time) > 3:
166. hrpart = "00"
167. minpart = time[0:2]
168. **else**:
169. hrpart = "00"
170. minpart = "0" + time[0:1]
171. convtime = hrpart[0:2] + ":" + minpart[0:2]
172. **return**(convtime)
174. # Apply the function to the Time column
175. df['Time'] = df['Time'].apply(converttime)
177. # Combine the FlightDate with the formatted Time Column in a specfic format
178. df['FlightDateTime'] = pd.to\_datetime(df['FlightDate'] + ' ' + df['Time'], format='%Y-%m-%d %H:%M')
180. # Remove the columns that are no longer required
181. df = df[['FlightDateTime','TaxiOut','Timeband']]
183. # Re-index the dataframe with the FlightDateTime
184. df.set\_index('FlightDateTime', inplace=True)
186. # In[16]:
188. df['TaxiOut'].plot()
189. plt.ylabel("TaxiOut")
190. plt.xlabel("FlightDateTime")
191. plt.show()

194. # In[17]:
196. df['TaxiOut'].describe()

199. # In[18]:
201. # NULLs
202. info=pd.DataFrame(df.isnull().sum()).T.rename(index={0:'# NULL'})
203. info=info.append(pd.DataFrame(df.isnull().sum()/df.shape[0]\*100).T.rename(index={0:'% NULL'}))
204. info=info.append(pd.DataFrame(df.dtypes).T.rename(index={0:'type'}))
205. info

208. # In[19]:
210. dfd = df[['TaxiOut']].resample("D").mean()
211. dfd['TaxiOut'].plot(color = "yellow", label = "Day")
213. dfw = df[['TaxiOut']].resample("W").mean()
214. dfw['TaxiOut'].plot(color = "red", label = "Week")
216. dfm = df[['TaxiOut']].resample("M").mean()
217. dfm['TaxiOut'].plot(color = "blue", label = "Month")
219. plt.ylabel("TaxiOut Mean")
220. plt.xlabel("Flight Date")
221. plt.legend(loc='best')
222. plt.show()
224. # #### Time bands
226. # In[20]:
228. # Show the count of flights departing in each Timeband
229. df.groupby('Timeband').agg({'Timeband':'count'}).plot(kind='bar',sort\_columns = 'Timeband')
230. plt.ylabel("Flight Count")
231. plt.legend(loc='best')
232. plt.show()

235. # In[21]:
237. plt.rcParams['figure.figsize'] = [8.0, 5.0]
238. dfw5am = df[df['Timeband'] == '05'][['TaxiOut']].resample("M").mean()
239. dfw5am['TaxiOut'].plot(color = "orange", label = "5am")
241. dfd1pm = df[df['Timeband'] == '13'][['TaxiOut']].resample("M").mean()
242. dfd1pm['TaxiOut'].plot(color = "black", label = "1pm")
244. dfm5pm = df[df['Timeband'] == '17'][['TaxiOut']].resample("M").mean()
245. dfm5pm['TaxiOut'].plot(color = "blue", label = "5pm")
247. dfm6pm = df[df['Timeband'] == '18'][['TaxiOut']].resample("M").mean()
248. dfm6pm['TaxiOut'].plot(color = "red", label = "6pm")
250. plt.ylabel("Taxi Out Mean")
251. plt.xlabel("Flight Date")
252. plt.legend(loc='best')
253. plt.show()
255. # In[22]:
257. decomposition = seasonal\_decompose(dfm.TaxiOut)
258. plt.rcParams['figure.figsize'] = [8.0, 10.0]
259. fig = decomposition.plot()
260. plt.show()

263. # In[23]:
265. #Calculate rolling mean
266. rolmean12 = dfm.rolling(window=12).mean()
268. # Create two subplots sharing y axis
269. plt.rcParams['figure.figsize'] = [8.0, 8.0]
270. fig, (ax1, ax2) = plt.subplots(2, sharey=True)
272. ax1.plot(rolmean12, color = 'black')
273. ax1.set(title='Rolling Mean vs Trend from seasonal\_decompose', ylabel='Rolling Mean')
275. ax2.plot(decomposition.trend, color = 'red')
276. ax2.set(xlabel='Date', ylabel='Trend')
278. plt.show()
280. # In[24]:
282. # Difference the monthly data
283. dfms = dfm.diff(1)
284. dfms12 = dfm.diff(12)
285. # Drop null values
286. dfms = dfms.dropna()
287. dfms12 = dfms12.dropna()
289. plt.rcParams['figure.figsize'] = [8.0, 5.0]
291. # Plot the differenced values
292. dfms['2016-03-01':'2017-06-01'].plot(color = "blue", title ="Differenced (1)")
293. plt.xlabel("Date")
294. plt.show()
296. # Plot the 12 differenced values
297. dfms12['2016-03-01':'2017-06-01'].plot(color = "red", title = "Differenced (12)")
298. plt.xlabel("Date")
299. plt.show()
301. #Plot the residual element of the seasonal\_decomposition function
302. decomposition.resid['2016-03-01':'2017-06-01'].plot(color = "black", title = "Residual")
303. plt.xlabel("Date")
304. plt.show()
306. # plot the autocorrelation of the differenced 1 values
307. plot\_acf(dfms.TaxiOut, lags=20)
309. # In[25]:
311. # Create function to run the Dickey-Fuller test
312. **def** TestAdfuller(ts, cutoff = 0.01):
313. ts\_test = adfuller(ts, autolag = 'AIC')
314. ts\_test\_output = pd.Series(ts\_test[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
315. **print**("p-value = ", ts\_test[1])
316. **print**("cutoff = ", cutoff,"\n")
318. **print**("\*\* Original data \*\*")
319. TestAdfuller(dfm['TaxiOut'])
321. **print**("\*\* Difference of lag 1 \*\*")
322. TestAdfuller(dfms['TaxiOut'])
324. **print**("\*\* Difference of lag 12 \*\*")
325. TestAdfuller(dfms12['TaxiOut'])

328. # In[26]:
330. # Split the data into train and test sets
331. train = dfm[0:24]
332. test = dfm[24:]

335. # In[27]:
337. # Plot pacf
338. plt.rcParams['figure.figsize'] = [8.0, 5.0]
339. plot\_pacf(train.TaxiOut, lags=20)

342. # In[58]:
344. # Fit the data to an AR model with a p value between 0 and 7 and store the resultant BIC
345. plt.rcParams['figure.figsize'] = [8.0, 5.0]
346. BIC = np.zeros(8)
347. **for** p **in** range(8):
348. mod = ARIMA(train, order=(p,0,0))
349. res = mod.fit()
350. # Save BIC
351. BIC[p] = res.bic
352. # Plot the BIC over p
353. plt.plot(range(1,8), BIC[1:8], marker='o', color = "blue")
354. plt.xlabel('AR(p)')
355. plt.ylabel('Bayesian Information Criterion')
356. plt.show()

359. # In[29]:
361. # Fit the data to an AR model with a p value between 0 and 7 and store the resultant AIC
362. plt.rcParams['figure.figsize'] = [8.0, 5.0]
363. AIC = np.zeros(8)
364. **for** p **in** range(8):
365. mod = ARIMA(train, order=(p,0,0))
366. res = mod.fit()
367. # Save AIC
368. AIC[p] = res.aic
369. # Plot the AIC over p
370. plt.plot(range(1,8), AIC[1:8], marker='o', color = "blue")
371. plt.xlabel('AR(p)')
372. plt.ylabel('Akaike Information Criterion')
373. plt.show()
375. # In[30]:
377. # Plot the acf function
378. plt.rcParams['figure.figsize'] = [8.0, 5.0]
379. plot\_acf(train.TaxiOut, lags=20)
380. plt.show()
382. autocorrelation = train['TaxiOut'].autocorr()
383. **print**("Autocorrelation of Taxi Out = ", autocorrelation)
385. # In[31]:
387. # AR(2) model, turning off the I and MA components of the ARIMA model with 0 d and q values.
388. p = 2
389. d = 0
390. q = 0
391. mod200 = ARIMA(train, order=(p,d,q))
392. res200 = mod200.fit()
394. # AR(3) model to see how this compares to AR(2), turning off the I and MA components of the ARIMA model with 0 d and q values.
395. p = 3
396. d = 0
397. q = 0
398. mod300 = ARIMA(train, order=(p,d,q))
399. res300 = mod300.fit()
401. # ARIMA(2,1,0) I will now add a differencing of 1 to the AR(2) model
402. p = 2
403. d = 1
404. q = 0
405. mod210 = ARIMA(train, order=(p,d,q))
406. res210 = mod210.fit()
408. # ARIMA(3,1,0) I will now add a differencing of 1 to the AR(2) model
409. p = 3
410. d = 1
411. q = 0
412. mod310 = ARIMA(train, order=(p,d,q))
413. res310 = mod310.fit()
415. # MA(1) model, turning off the AR and I components of the ARIMA model with 0 value p and d values.
416. p = 0
417. d = 0
418. q = 1
419. mod001 = ARIMA(train, order=(p,d,q))
420. res001 = mod001.fit()
422. # ARIMA(0,1,1) I will now add a differencing of 1 to the MA(1) model
423. p = 0
424. d = 1
425. q = 1
426. mod011 = ARIMA(train, order=(p,d,q))
427. res011 = mod011.fit()
429. # ARIMA(2,1,1)
430. p = 2
431. d = 1
432. q = 1
433. mod211 = ARIMA(train, order=(p,d,q))
434. res211 = mod211.fit()
436. # ARIMA(3,1,1)
437. p = 3
438. d = 1
439. q = 1
440. mod311 = ARIMA(train, order=(p,d,q))
441. res311 = mod311.fit()
443. # In[32]:
445. # Define the prediction start and end months
446. predstart = 1
447. predend = 28
448. plt.rcParams['figure.figsize'] = [16.0, 20.0]
450. ax1 = plt.subplot(4, 2, 1)
451. ax2 = plt.subplot(4, 2, 2)
452. ax3 = plt.subplot(4, 2, 3)
453. ax4 = plt.subplot(4, 2, 4)
454. ax5 = plt.subplot(4, 2, 5)
455. ax6 = plt.subplot(4, 2, 6)
456. ax7 = plt.subplot(4, 2, 7)
457. ax8 = plt.subplot(4, 2, 8)
458. # AR(2)
459. fig = res200.plot\_predict(start=predstart, end=predend, ax = ax1)
460. ax1.set\_title("AR(2) Model Prediction")
462. # AR(3)
463. res300.plot\_predict(start=predstart, end=predend, ax = ax2)
464. ax2.set\_title("AR(3) Model Prediction")
466. # ARIMA(2,1,0)
467. res210.plot\_predict(start=predstart, end=predend, ax = ax3)
468. ax3.set\_title("ARIMA(2,1,0) Model Prediction")
470. # ARIMA(3,1,0)
471. res310.plot\_predict(start=predstart, end=predend, ax = ax4)
472. ax4.set\_title("ARIMA(3,1,0) Model Prediction")
474. # MA(1)
475. res001.plot\_predict(start=predstart, end=predend, ax = ax5)
476. ax5.set\_title("MA(1) Model Prediction")
478. # ARIMA(0,1,1)
479. res011.plot\_predict(start=predstart, end=predend, ax = ax6)
480. ax6.set\_title("ARIMA(0,1,1) Model Prediction")
482. # ARIMA(2,1,1)
483. res211.plot\_predict(start=predstart, end=predend, ax = ax7)
484. ax7.set\_title("ARIMA(2,1,1) Model Prediction")
486. # ARIMA(3,1,1)
487. res311.plot\_predict(start=predstart, end=predend, ax = ax8)
488. ax8.set\_title("ARIMA(3,1,1) Model Prediction")
489. plt.show()

492. # In[33]:
494. plt.rcParams['figure.figsize'] = [15.0, 35.0]
495. ax1 = plt.subplot(8, 2, 1)
496. ax2 = plt.subplot(8, 2, 2)
497. ax3 = plt.subplot(8, 2, 3)
498. ax4 = plt.subplot(8, 2, 4)
499. ax5 = plt.subplot(8, 2, 5)
500. ax6 = plt.subplot(8, 2, 6)
501. ax7 = plt.subplot(8, 2, 7)
502. ax8 = plt.subplot(8, 2, 8)
503. ax9 = plt.subplot(8, 2, 9)
504. ax10 = plt.subplot(8, 2, 10)
505. ax11 = plt.subplot(8, 2, 11)
506. ax12 = plt.subplot(8, 2, 12)
507. ax13 = plt.subplot(8, 2, 13)
508. ax14 = plt.subplot(8, 2, 14)
509. ax15 = plt.subplot(8, 2, 15)
510. ax16 = plt.subplot(8, 2, 16)
512. # AR(2)
513. residuals = pd.DataFrame(res200.resid)
514. residuals.plot(title = "AR(2) Residuals",kind='density', ax = ax1)
515. plot\_acf(residuals,title = "AR(2) Autocorrelation - Residuals", ax = ax2)
517. # AR(3)
518. residuals = pd.DataFrame(res300.resid)
519. residuals.plot(title = "AR(3) Residuals",kind='density', ax = ax3)
520. plot\_acf(residuals,title = "AR(3) Autocorrelation - Residuals", ax = ax4)
522. # ARIMA(2,1,0)
523. residuals = pd.DataFrame(res210.resid)
524. residuals.plot(title = "ARIMA(2,1,0) Residuals",kind='density', ax = ax5)
525. plot\_acf(residuals,title = "ARIMA(2,1,0) Autocorrelation - Residuals", ax = ax6)
527. # ARIMA(3,1,0)
528. residuals = pd.DataFrame(res310.resid)
529. residuals.plot(title = "ARIMA(3,1,0) Residuals",kind='density', ax = ax7)
530. plot\_acf(residuals,title = "ARIMA(3,1,0) Autocorrelation - Residuals)", ax = ax8)
532. # MA(1)
533. residuals = pd.DataFrame(res001.resid)
534. residuals.plot(title = "MA(1) Residuals",kind='density', ax = ax9)
535. plot\_acf(residuals,title = "MA(1) Autocorrelation - Residuals", ax = ax10)
537. # ARIMA(0,1,1)
538. residuals = pd.DataFrame(res011.resid)
539. residuals.plot(title = "ARIMA(0,1,1) Residuals",kind='density', ax = ax11)
540. plot\_acf(residuals, title = "ARIMA(0,1,1) Autocorrelation - Residuals", ax = ax12)
542. # ARIMA(2,1,1)
543. residuals = pd.DataFrame(res211.resid)
544. residuals.plot(title = "ARIMA(2,1,1) Residuals", kind='density', ax = ax13)
545. plot\_acf(residuals, title = "ARIMA(2,1,1) Autocorrelation - Residuals", ax = ax14)
547. # ARIMA(3,1,1)
548. residuals = pd.DataFrame(res311.resid)
549. residuals.plot(title = "ARIMA(3,1,1) Residuals", kind='density', ax = ax15)
550. plot\_acf(residuals, title = "ARIMA(3,1,1) Autocorrelation - Residuals", ax = ax16)
551. plt.show()
553. # In[34]:
555. resdf = pd.DataFrame([res200.params[0], res200.params[1], res200.aic, res200.bic]).T.rename(index={0:'AR(2)'})
556. resdf = resdf.append(pd.DataFrame([res300.params[0], res300.params[1], res300.aic, res300.bic]).T.rename(index={0:'AR(3)'}))
557. resdf = resdf.append(pd.DataFrame([res210.params[0], res210.params[1], res210.aic, res210.bic]).T.rename(index={0:'ARIMA(2,1,0)'}))
558. resdf = resdf.append(pd.DataFrame([res310.params[0], res310.params[1], res310.aic, res310.bic]).T.rename(index={0:'ARIMA(3,1,0)'}))
559. resdf = resdf.append(pd.DataFrame([res001.params[0], res001.params[1], res001.aic, res001.bic]).T.rename(index={0:'MA(1)'}))
560. resdf = resdf.append(pd.DataFrame([res211.params[0], res211.params[1], res211.aic, res211.bic]).T.rename(index={0:'ARIMA(2,1,1)'}))
561. resdf = resdf.append(pd.DataFrame([res311.params[0], res311.params[1], res311.aic, res311.bic]).T.rename(index={0:'ARIMA(3,1,1)'}))
563. resdf = resdf.rename(columns={0:'μ',1:'ϕ',2:'AIC',3:'BIC'})
564. resdf