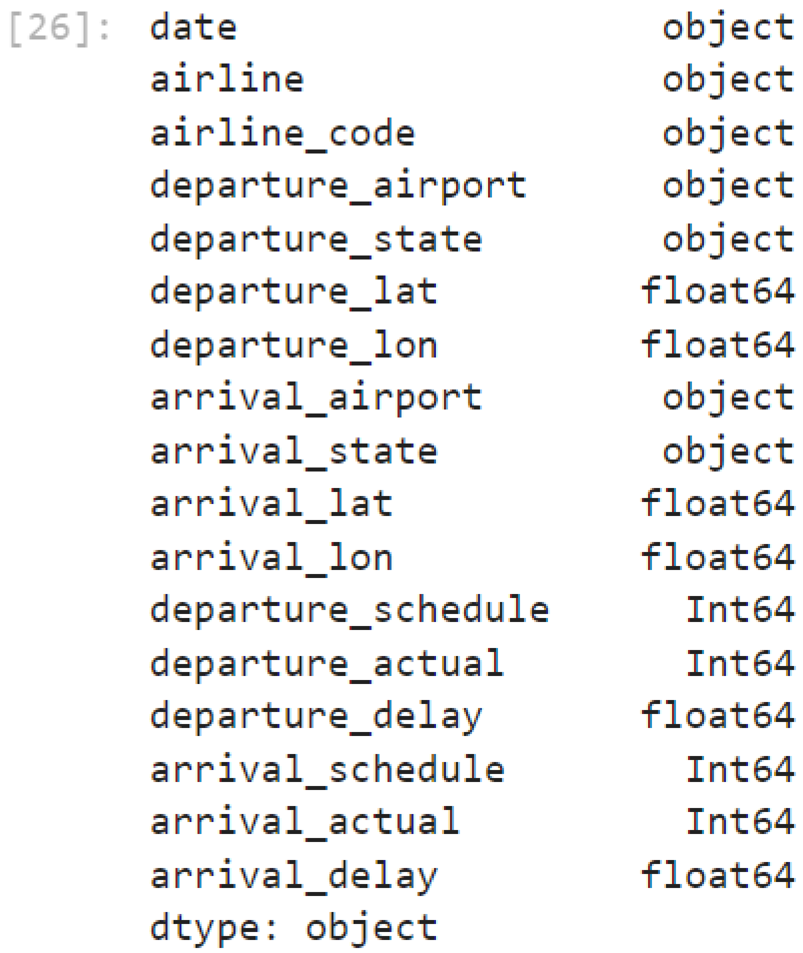
Predict Flight Delay based on Linear Regression

Data Description

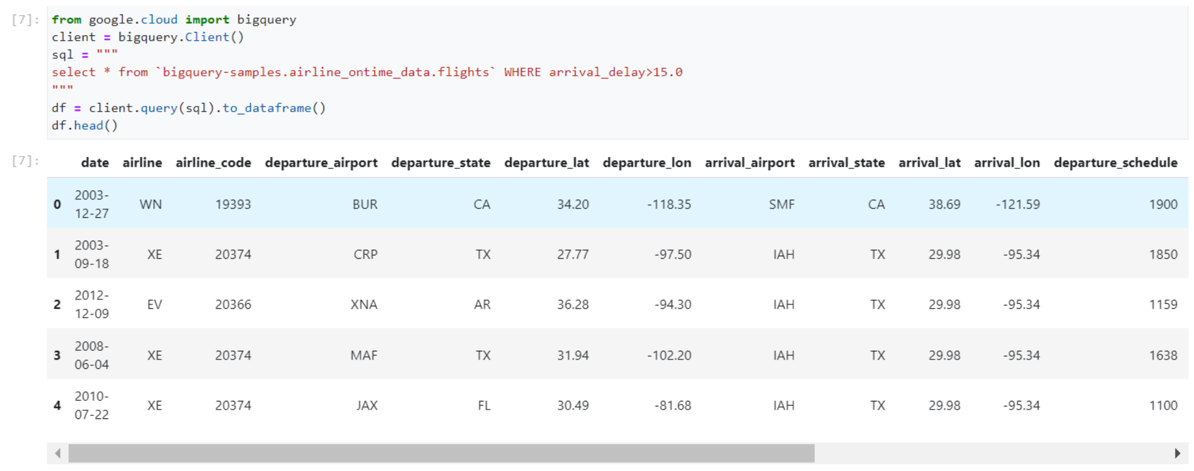
The data set used for analysis contains data about flights leaving from 24 airlines and 344 arrival airport and 347 departure airport between 2002 to 2012. It was obtained from open data at `bigquery-samples.airline_ontime_data.flights`. It includes 70588485 lines of individual flight information with 17 columns and flight id-codes are also included from `bigquery-samples.airline_ontime_data.airline_id_codes`.



```
[26]: date                      object
      airline                   object
      airline_code              object
      departure_airport         object
      departure_state           object
      departure_lat            float64
      departure_lon            float64
      arrival_airport           object
      arrival_state             object
      arrival_lat              float64
      arrival_lon              float64
      departure_schedule         Int64
      departure_actual           Int64
      departure_delay          float64
      arrival_schedule           Int64
      arrival_actual             Int64
      arrival_delay            float64
      dtype: object
```

Exploratory Data Analysis (EDA)

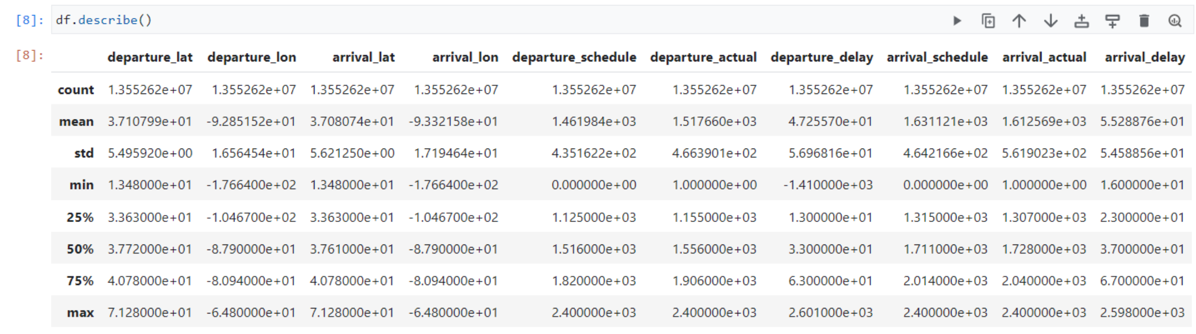
A flight is on-time if the arrival delay is within 15-min of the scheduled arrival time (CRSDepTime). A flight is delayed if the arrival delay is more than 15-min late from the scheduled arrival time (CRSDepTime). We would like to build an analysis dataset by choosing the threshold of 15 minutes, beyond which we consider the class change to "delayed" flight. This is a standard threshold in the aviation industry, with indicators on delayed flights commonly based on 15 minutes of delay. Thus, I will just keep the flights with arrival delay greater than 15.



```
[7]: from google.cloud import bigquery
     client = bigquery.Client()
     sql = """
     select * from `bigquery-samples.airline_ontime_data.flights` WHERE arrival_delay>15.0
     """
     df = client.query(sql).to_dataframe()
     df.head()
```

[7]:

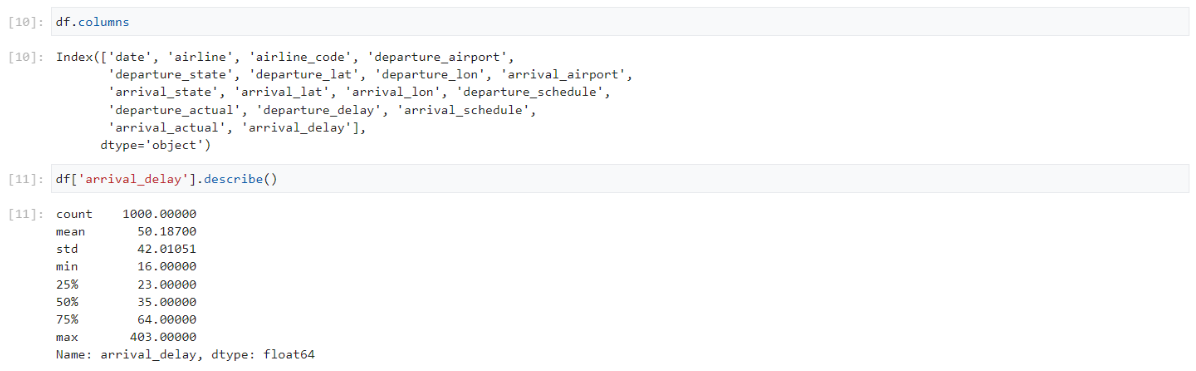
	date	airline	airline_code	departure_airport	departure_state	departure_lat	departure_lon	arrival_airport	arrival_state	arrival_lat	arrival_lon	departure_schedule
0	2003-12-27	WN	19393	BUR	CA	34.20	-118.35	SMF	CA	38.69	-121.59	1900
1	2003-09-18	XE	20374	CRP	TX	27.77	-97.50	IAH	TX	29.98	-95.34	1850
2	2012-12-09	EV	20366	XNA	AR	36.28	-94.30	IAH	TX	29.98	-95.34	1159
3	2008-06-04	XE	20374	MAF	TX	31.94	-102.20	IAH	TX	29.98	-95.34	1638
4	2010-07-22	XE	20374	JAX	FL	30.49	-81.68	IAH	TX	29.98	-95.34	1100



```
[8]: df.describe()
```

[8]:

	departure_lat	departure_lon	arrival_lat	arrival_lon	departure_schedule	departure_actual	departure_delay	arrival_schedule	arrival_actual	arrival_delay
count	1.355262e+07	1.355262e+07	1.355262e+07	1.355262e+07	1.355262e+07	1.355262e+07	1.355262e+07	1.355262e+07	1.355262e+07	1.355262e+07
mean	3.710799e+01	-9.285152e+01	3.708074e+01	-9.332158e+01	1.461984e+03	1.517660e+03	4.725570e+01	1.631121e+03	1.612569e+03	5.528876e+01
std	5.495920e+00	1.656454e+01	5.621250e+00	1.719464e+01	4.351622e+02	4.663901e+02	5.696816e+01	4.642166e+02	5.619023e+02	5.458856e+01
min	1.348000e+01	-1.766400e+02	1.348000e+01	-1.766400e+02	0.000000e+00	1.000000e+00	-1.410000e+03	0.000000e+00	1.000000e+00	1.600000e+01
25%	3.363000e+01	-1.046700e+02	3.363000e+01	-1.046700e+02	1.125000e+03	1.155000e+03	1.300000e+01	1.315000e+03	1.307000e+03	2.300000e+01
50%	3.772000e+01	-8.790000e+01	3.761000e+01	-8.790000e+01	1.516000e+03	1.556000e+03	3.300000e+01	1.711000e+03	1.728000e+03	3.700000e+01
75%	4.078000e+01	-8.094000e+01	4.078000e+01	-8.094000e+01	1.820000e+03	1.906000e+03	6.300000e+01	2.014000e+03	2.040000e+03	6.700000e+01
max	7.128000e+01	-6.480000e+01	7.128000e+01	-6.480000e+01	2.400000e+03	2.400000e+03	2.601000e+03	2.400000e+03	2.400000e+03	2.598000e+03



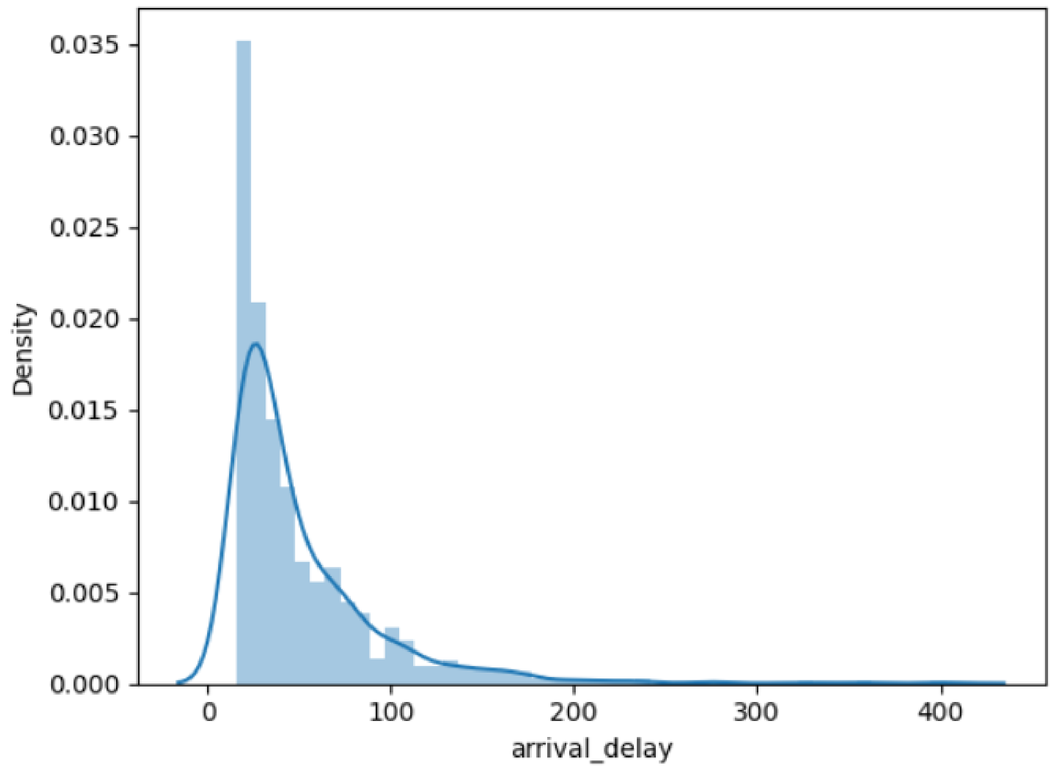
```
[10]: df.columns

[10]: Index(['date', 'airline', 'airline_code', 'departure_airport',
             'departure_state', 'departure_lat', 'departure_lon', 'arrival_airport',
             'arrival_state', 'arrival_lat', 'arrival_lon', 'departure_schedule',
             'departure_actual', 'departure_delay', 'arrival_schedule',
             'arrival_actual', 'arrival_delay'],
            dtype='object')

[11]: df['arrival_delay'].describe()

[11]: count    1000.00000
      mean       50.18700
      std        42.01051
      min        16.00000
      25%        23.00000
      50%        35.00000
      75%        64.00000
      max       403.00000
      Name: arrival_delay, dtype: float64
```

Arrival Delay Distribution

The x-axis for the plot is to scale and as a result, we can see that the arrival delay distribution, leans toward left.

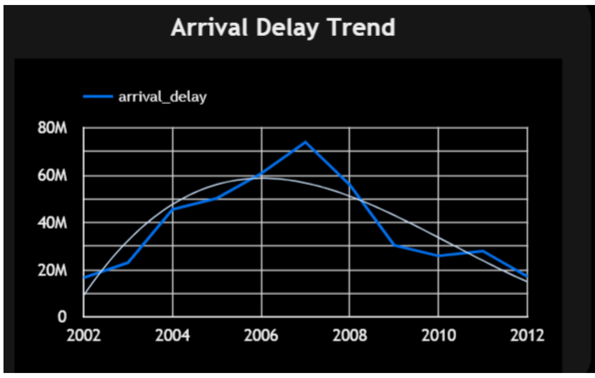
```
[12]: import pandas as pd
      import matplotlib.pyplot as plt
      import seaborn as sns
      import numpy as np
      from scipy.stats import norm
      from sklearn.preprocessing import StandardScaler
      from scipy import stats
      import warnings
      warnings.filterwarnings('ignore')
      %matplotlib inline
```

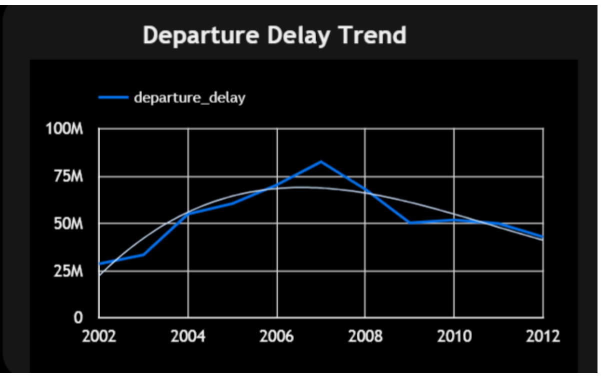
```
[13]: sns.distplot(df['arrival_delay']);
```



Average Departure & Arrival Delay by Year

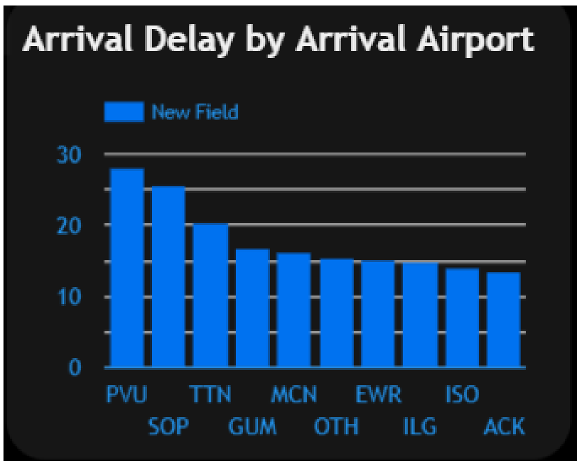
Next, we consider the impact of the years on the delays. A column chart with departure and arrival delay in minutes plotted by year is the most effective way to see the potential effects of the years.

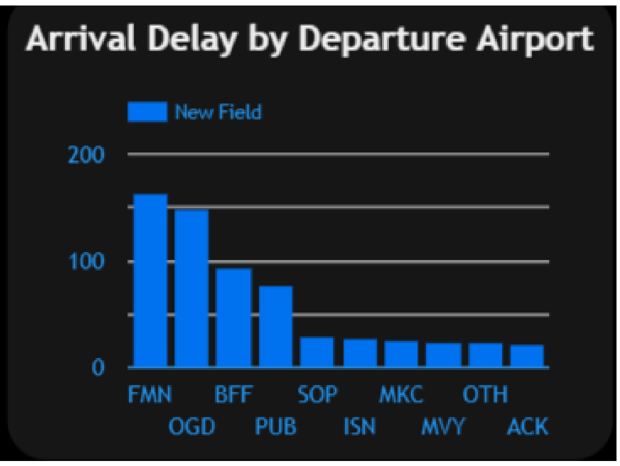


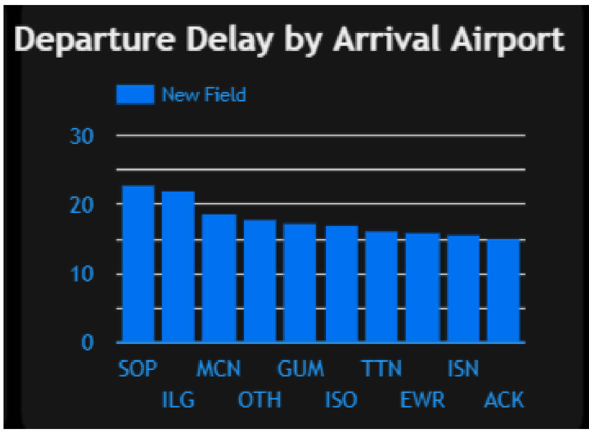


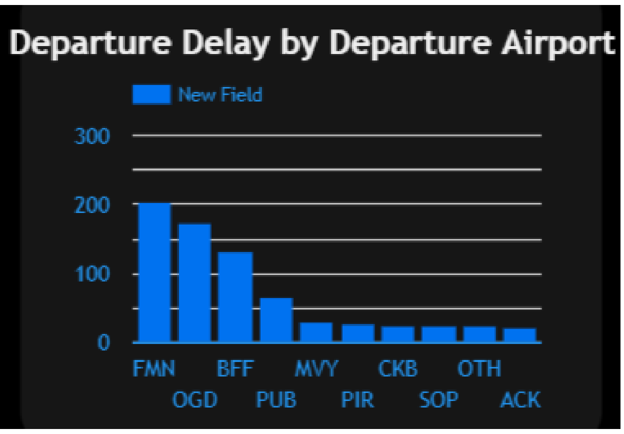
Average Arrival and Departure delay by Airports

Next, we consider average Arrival and Departure delay by airports and we can see a few trends as arrival and departure delay by departure delay has some common airports like FMN, OGD, BFF and PUB but there is not significant difference in arrival airports.



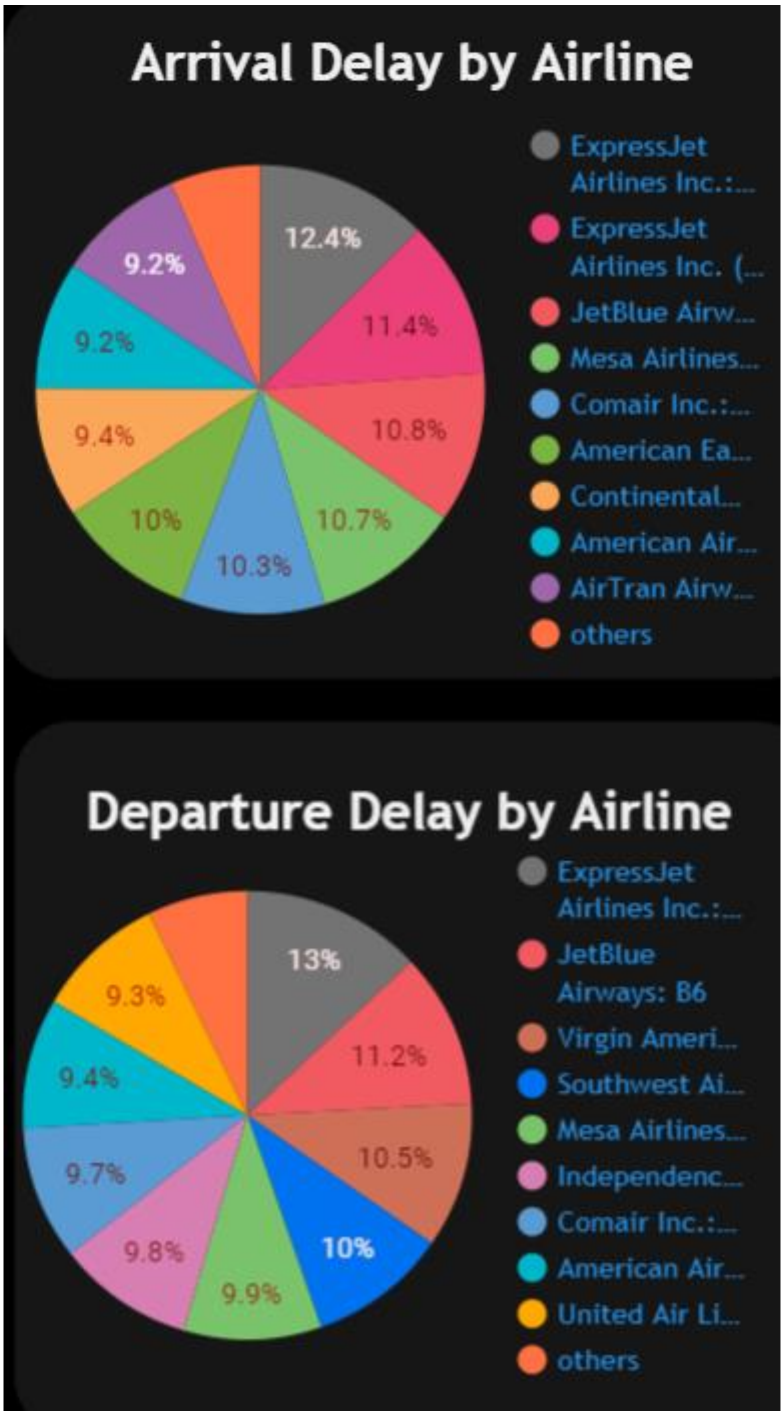






Arrival and Departure delay by Airline

We can see from the graph below that ExpressJet Airlines and JetBlue Airlines cause major percentage of delays in both arrival and departure.



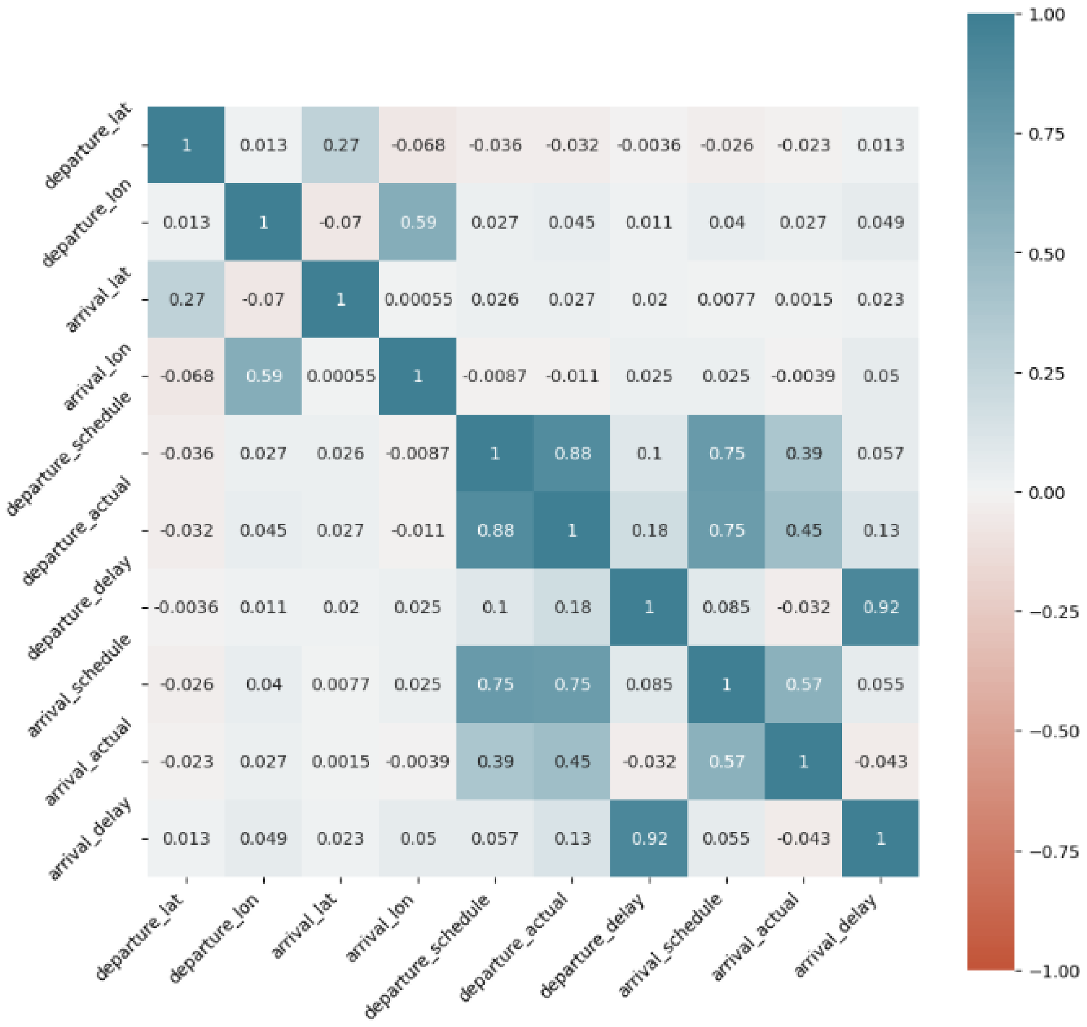
Correlation Analysis

Finally, before moving on to the modelling stage, I wanted to observe the correlation between the rows of the data frame. Below is the code I used to plot the correlation matrix. It showed that most columns had a relatively low (near zero) correlation with arrival delay, with the exception of the departure delay, which is expected. Though usually highly correlated columns are removed, I decided to include it as I think there is value in predicting the arrival delay when knowing the departure has been delayed.

From the below correlation coefficient heat map we can concur that for flight arrival delay prediction, the following features are potential candidates for the model:

1. Departure Longitude
2. Arrival Longitude
3. Departure Delay

[24]:
```python
plt.figure(figsize=(10,10))
corr = df.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True, annot=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
)
ax.set_yticklabels(
    ax.get_yticklabels(),
    rotation=45,

);
```



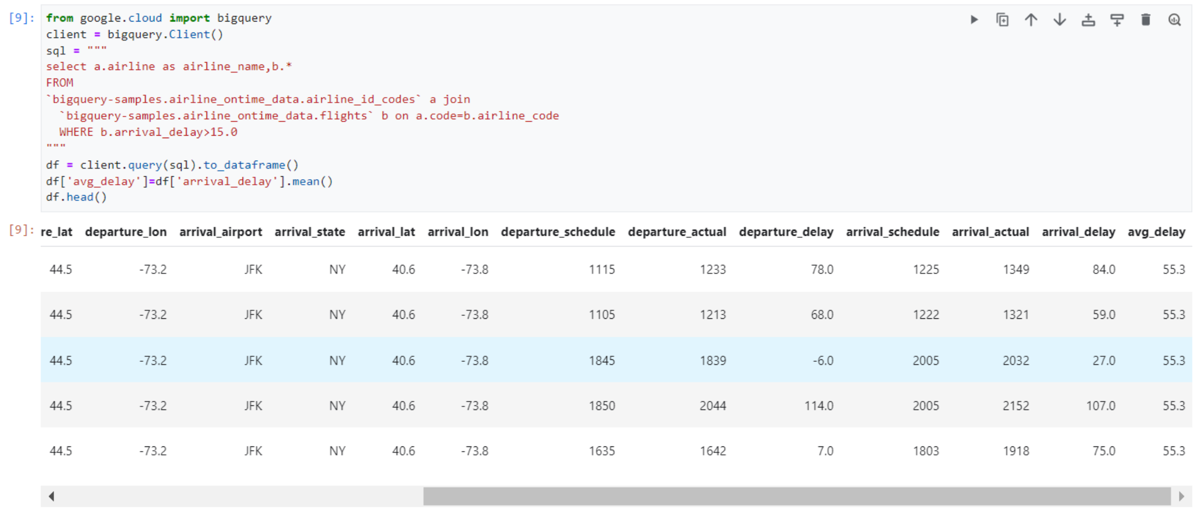
Data Modelling

Baseline Model

We begin by constructing a baseline model that we can use to compare to the other models. The baseline model I chose to use was simply to predict each flight's arrival delay to be the average arrival delay of its airline.

The avg_delay is already a column as we calculated the average delay of each airline then performed an inner join of this on our combined data frame during the EDA.

Next, we can use this as a predictor and evaluate the data loss and RMSE on the dataset.



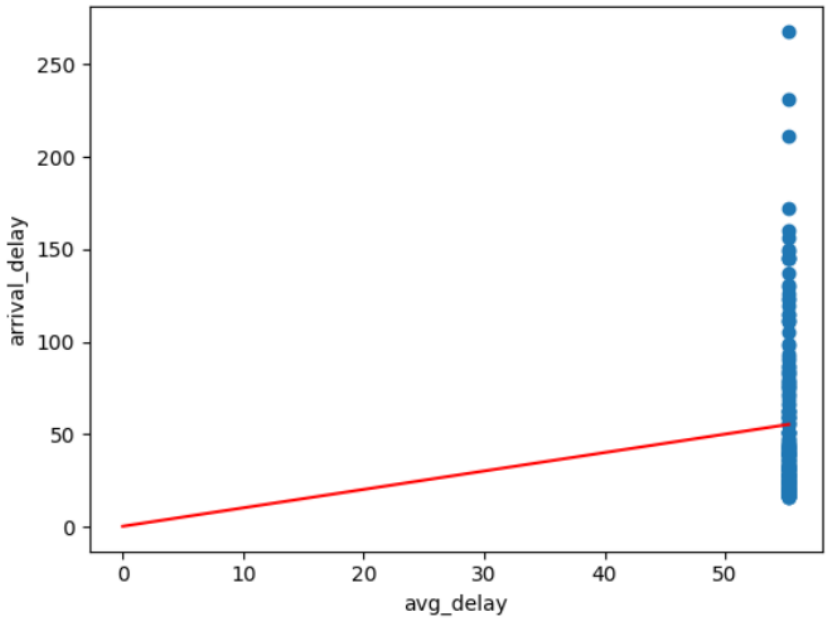
```
[9]: from google.cloud import bigquery
client = bigquery.Client()
sql = """
select a.airline as airline_name,b.*
FROM
`bigquery-samples.airline_ontime_data.airline_id_codes` a join
  `bigquery-samples.airline_ontime_data.flights` b on a.code=b.airline_code
  WHERE b.arrival_delay>15.0
"""
df = client.query(sql).to_dataframe()
df['avg_delay']=df['arrival_delay'].mean()
df.head()
```

[9]:

re_lat	departure_lon	arrival_airport	arrival_state	arrival_lat	arrival_lon	departure_schedule	departure_actual	departure_delay	arrival_schedule	arrival_actual	arrival_delay	avg_delay
44.5	-73.2	JFK	NY	40.6	-73.8	1115	1233	78.0	1225	1349	84.0	55.3
44.5	-73.2	JFK	NY	40.6	-73.8	1105	1213	68.0	1222	1321	59.0	55.3
44.5	-73.2	JFK	NY	40.6	-73.8	1845	1839	-6.0	2005	2032	27.0	55.3
44.5	-73.2	JFK	NY	40.6	-73.8	1850	2044	114.0	2005	2152	107.0	55.3
44.5	-73.2	JFK	NY	40.6	-73.8	1635	1642	7.0	1803	1918	75.0	55.3

Clearly, this is not a great predictor, as seen from the loss and RMSE. The RMSE is approximately the same as the standard deviation of the data, showing that it is a poor predictor. We will now use this as baseline for the following model.

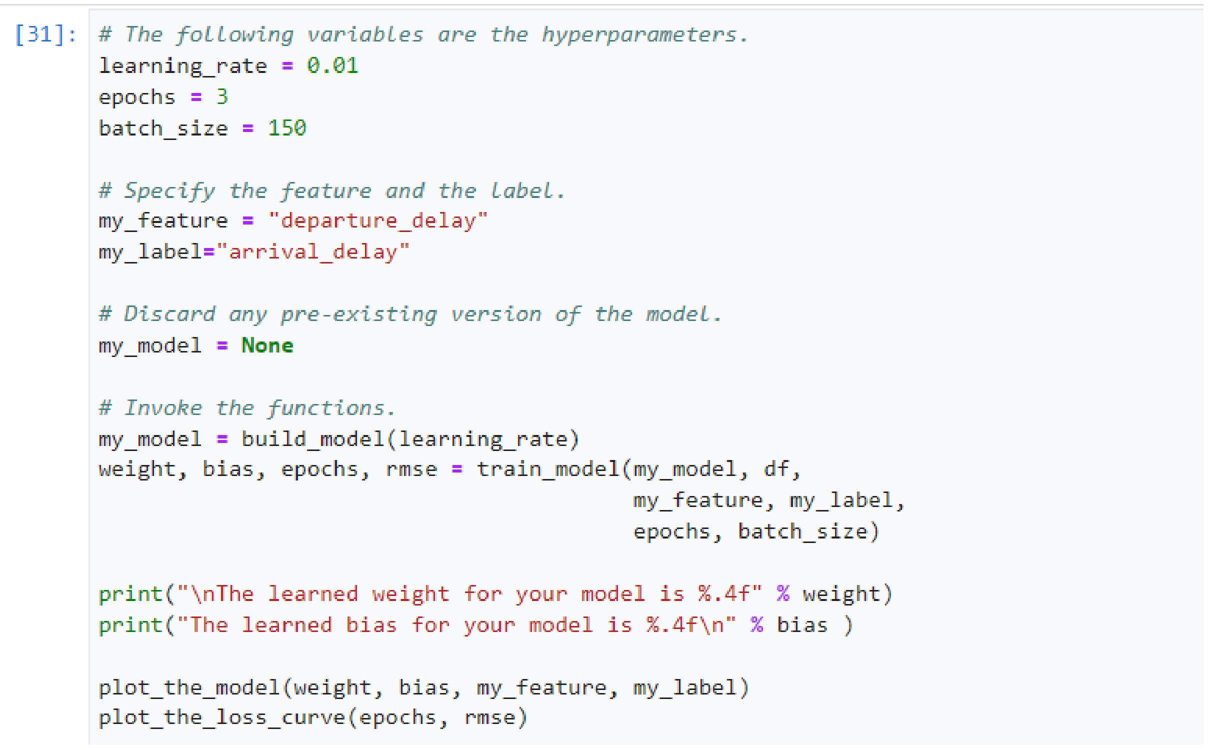
```
54211/54211 [==============================] - 96s 2ms/step - loss: 2980.9241 - root_mean_squared_error: 54.5978

The learned weight for your model is 0.9961
The learned bias for your model is 0.3186
```



Linear Regression

Next, I decided to use a linear regression model given that we want to output a continuous value (delay in minutes). I also chose to use it as I hypothesized that there may be a linear or more simplistic relationship between the features and labels; therefore, a linear regression could be a suitable model.

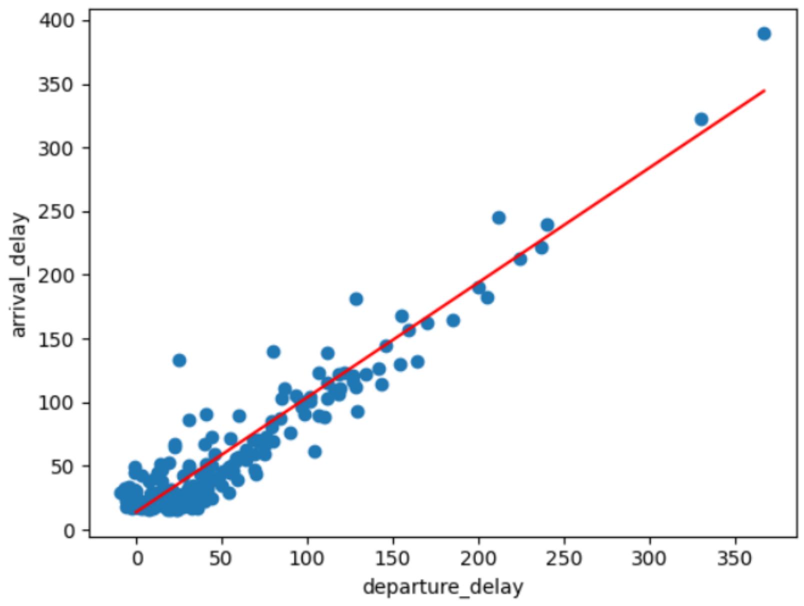
Using departure_delay as feature as it was the most correlated data row



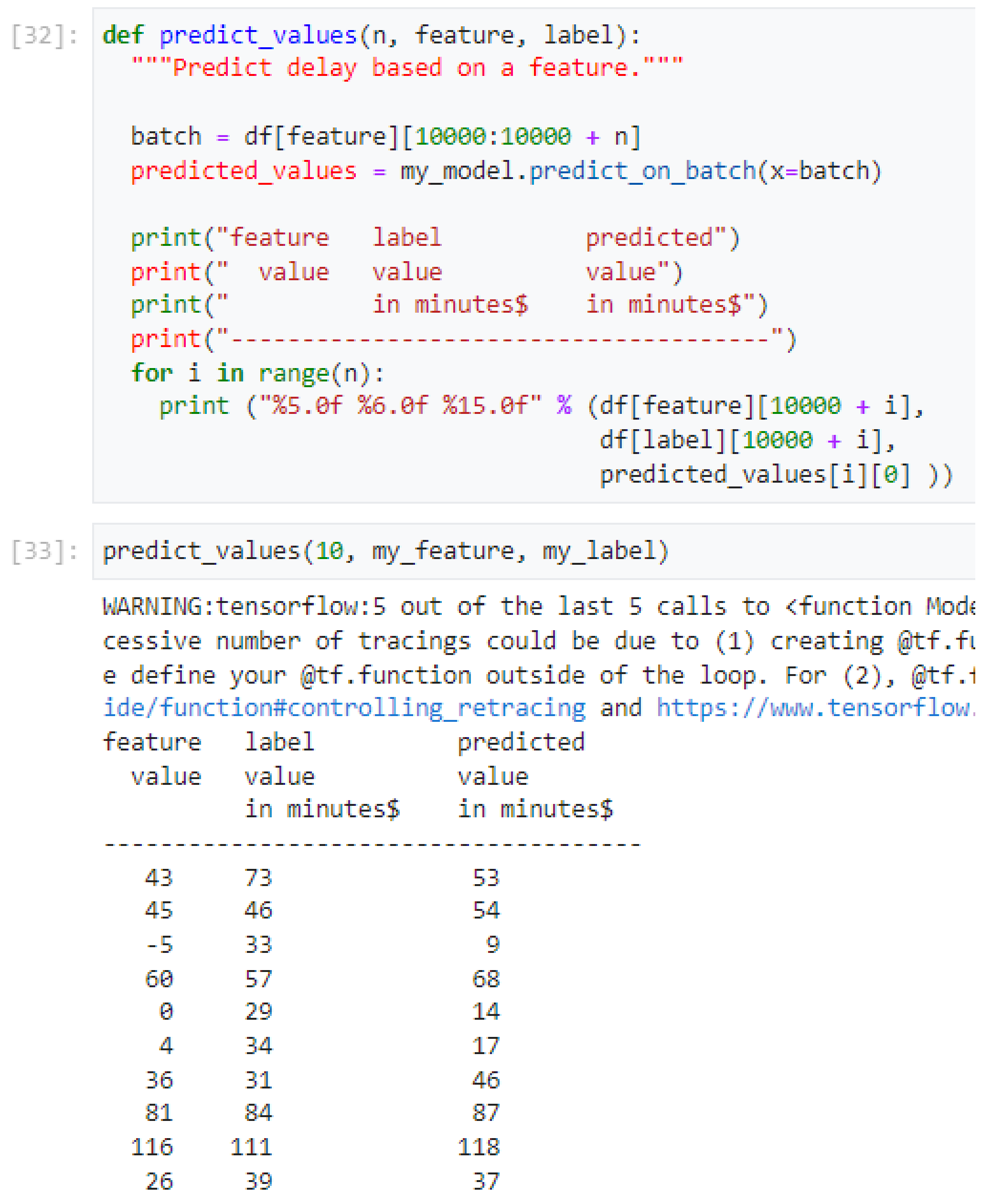
```
[31]: # The following variables are the hyperparameters.
      learning_rate = 0.01
      epochs = 3
      batch_size = 150

      # Specify the feature and the label.
      my_feature = "departure_delay"
      my_label="arrival_delay"

      # Discard any pre-existing version of the model.
      my_model = None

      # Invoke the functions.
      my_model = build_model(learning_rate)
      weight, bias, epochs, rmse = train_model(my_model, df,
                                               my_feature, my_label,
                                               epochs, batch_size)

      print("\nThe learned weight for your model is %.4f" % weight)
      print("The learned bias for your model is %.4f\n" % bias )

      plot_the_model(weight, bias, my_feature, my_label)
      plot_the_loss_curve(epochs, rmse)
```

We can see that loss is reduced to 459 and RMSE is reduced to 21 but we can still do better.

```
Epoch 1/3
90351/90351 [==============================] - 157s 2ms/step - loss: 459.5589 - root_mean_squared_error: 21.4373
Epoch 2/3
90351/90351 [==============================] - 157s 2ms/step - loss: 457.2025 - root_mean_squared_error: 21.3823
Epoch 3/3
90351/90351 [==============================] - 158s 2ms/step - loss: 457.1914 - root_mean_squared_error: 21.3820

The learned weight for your model is 0.9005
The learned bias for your model is 13.8253
```



Testing this to predict arrival_delay



```
[32]: def predict_values(n, feature, label):
        """Predict delay based on a feature."""

        batch = df[feature][10000:10000 + n]
        predicted_values = my_model.predict_on_batch(x=batch)

        print("feature   label          predicted")
        print("  value   value          value")
        print("          in minutes$    in minutes$")
        print("--------------------------------------")
        for i in range(n):
          print ("%5.0f %6.0f %15.0f" % (df[feature][10000 + i],
                                         df[label][10000 + i],
                                         predicted_values[i][0] ))
```

```
[33]: predict_values(10, my_feature, my_label)
```

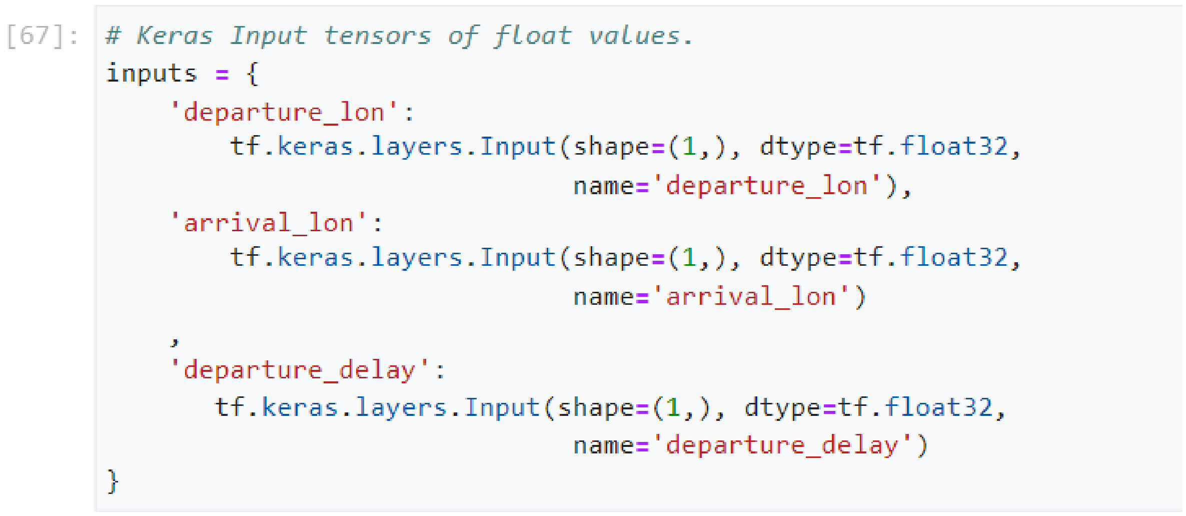
```
WARNING:tensorflow:5 out of the last 5 calls to <function Mode
cessive number of tracings could be due to (1) creating @tf.fu
e define your @tf.function outside of the loop. For (2), @tf.f
ide/function#controlling_retracing and https://www.tensorflow.
feature   label          predicted
  value   value          value
          in minutes$    in minutes$
--------------------------------------
   43     73              53
   45     46              54
   -5     33               9
   60     57              68
    0     29              14
    4     34              17
   36     31              46
   81     84              87
  116    111             118
   26     39              37
```

Using feature-cross

Using Feature-cross we can train our dataset on three features using **Input layers**.

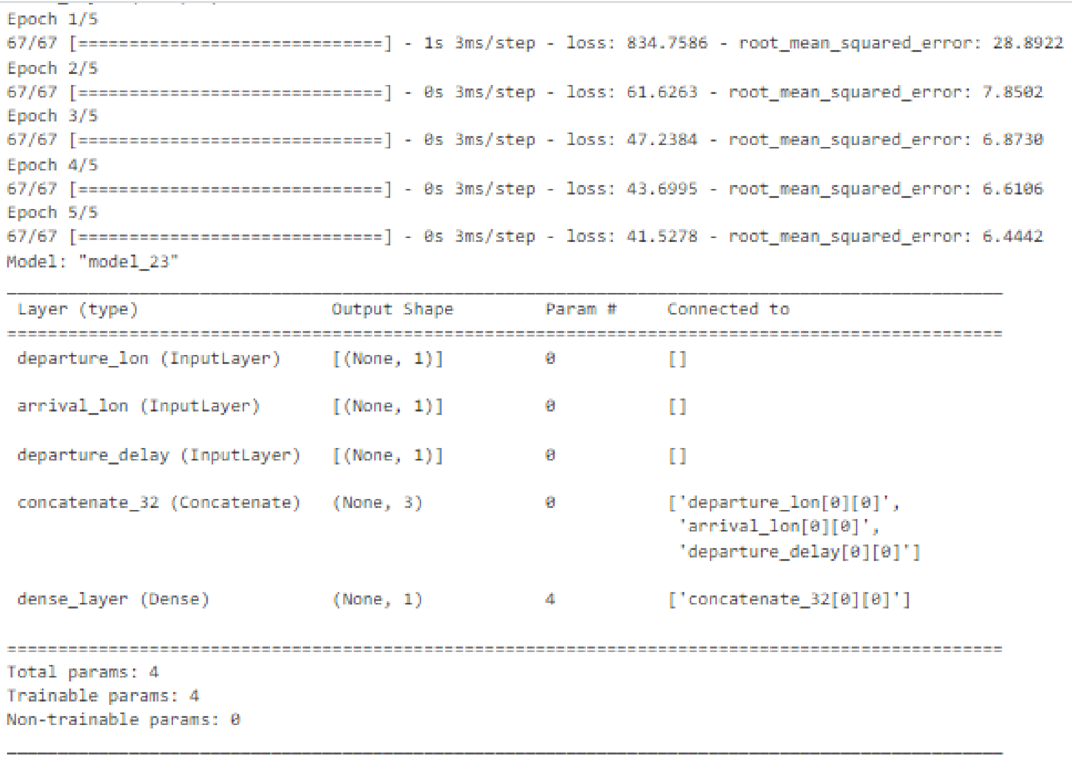
The following code cell defines three tf.keras.Input layers, one to represent departure_lon, one to represent arrival_lon and one for departure_delay all as floating-point values.

This code cell specifies the features that we'll ultimately train the model on and how each of those features will be represented.

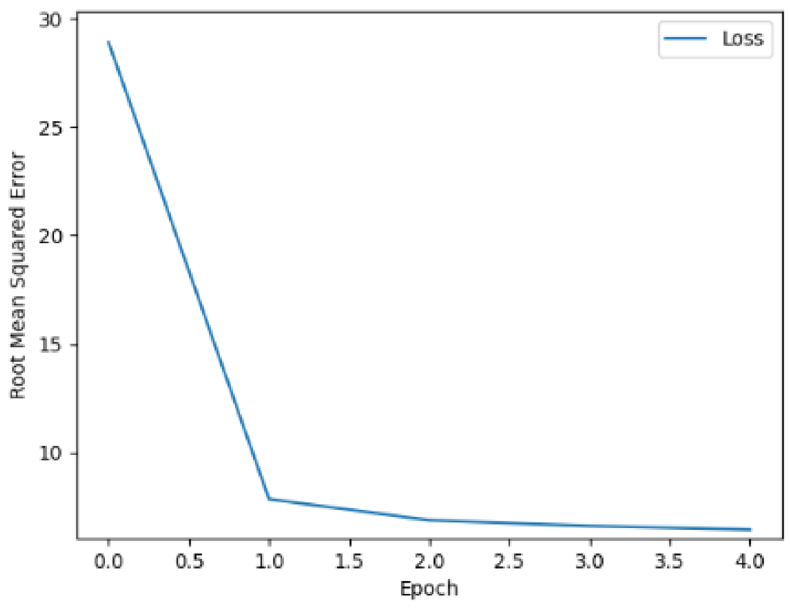


```
[67]: # Keras Input tensors of float values.
      inputs = {
          'departure_lon':
              tf.keras.layers.Input(shape=(1,), dtype=tf.float32,
                                    name='departure_lon'),
          'arrival_lon':
              tf.keras.layers.Input(shape=(1,), dtype=tf.float32,
                                    name='arrival_lon')
          ,
          'departure_delay':
             tf.keras.layers.Input(shape=(1,), dtype=tf.float32,
                                    name='departure_delay')
      }
```

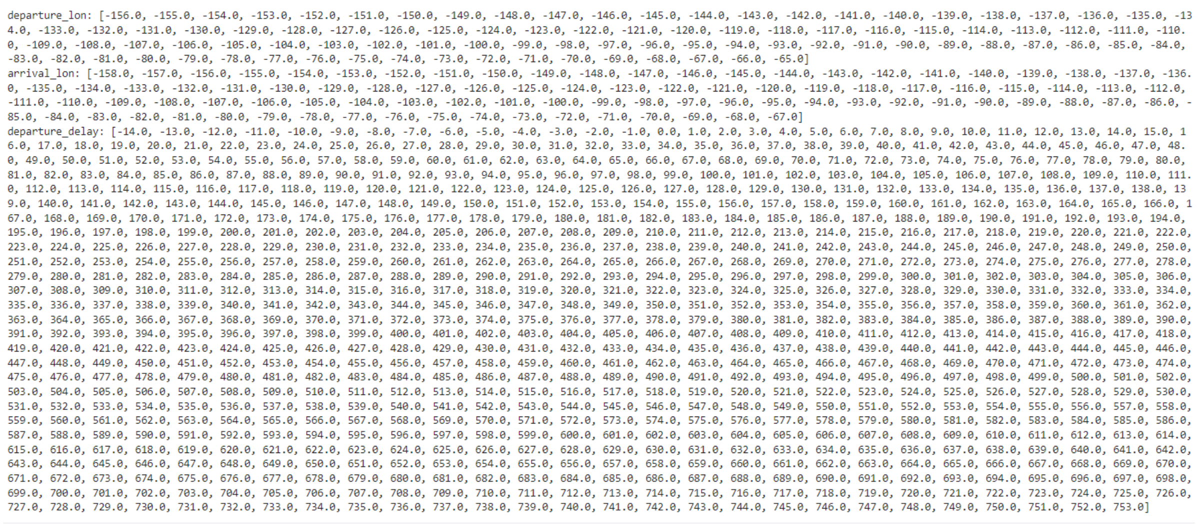
We can see that data loss is significantly reduced to 41 and RMSE is reduced to 6 but we can still do some optimizations as longitude and departure_delay as floating-point values does not have much predictive power. For example, flights originating from longitude 35 have not 36/35 less delay (or 35/36more delay) than flights originating from longitude 36.



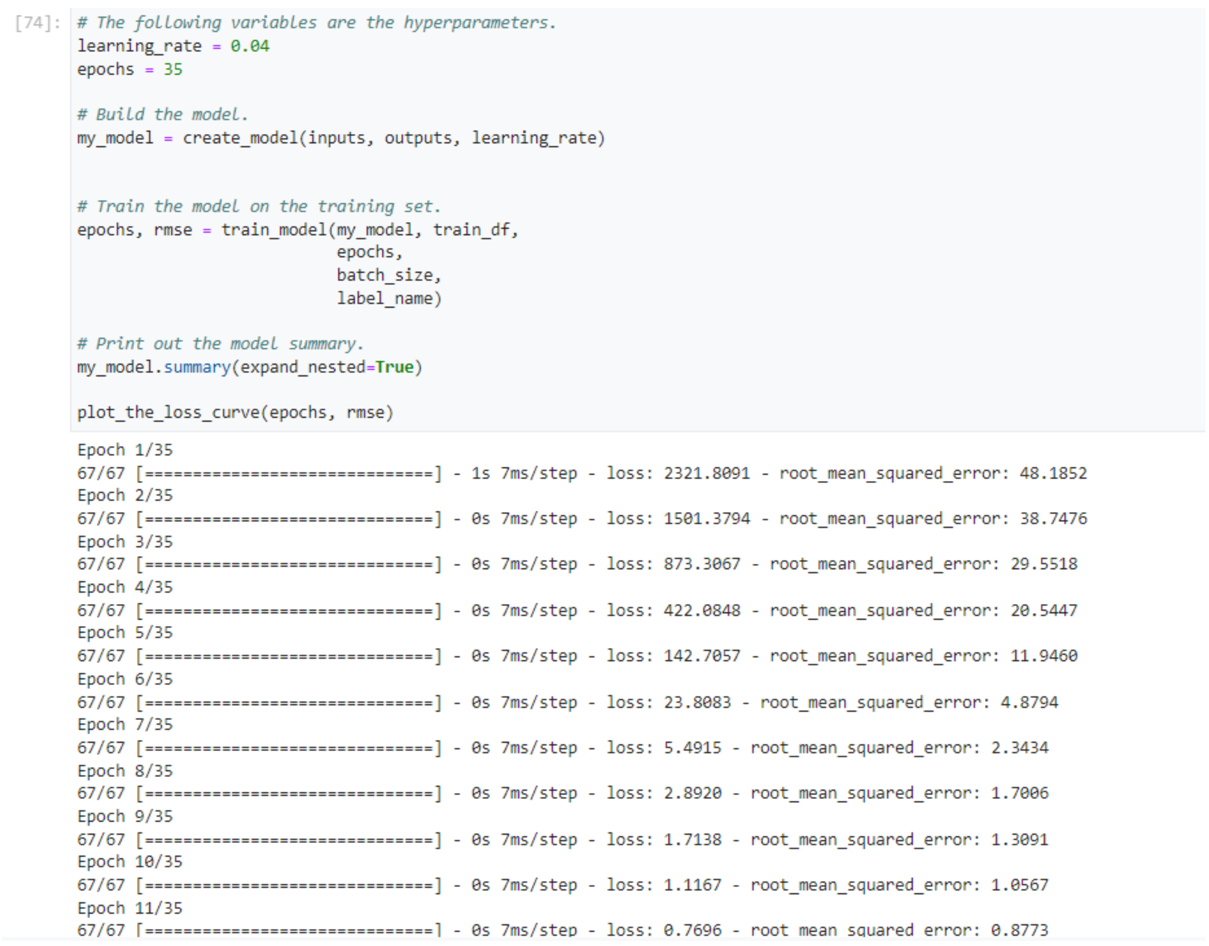
```
Epoch 1/5
67/67 [==============================] - 1s 3ms/step - loss: 834.7586 - root_mean_squared_error: 28.8922
Epoch 2/5
67/67 [==============================] - 0s 3ms/step - loss: 61.6263 - root_mean_squared_error: 7.8502
Epoch 3/5
67/67 [==============================] - 0s 3ms/step - loss: 47.2384 - root_mean_squared_error: 6.8730
Epoch 4/5
67/67 [==============================] - 0s 3ms/step - loss: 43.6995 - root_mean_squared_error: 6.6106
Epoch 5/5
67/67 [==============================] - 0s 3ms/step - loss: 41.5278 - root_mean_squared_error: 6.4442
Model: "model_23"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 departure_lon (InputLayer)     [(None, 1)]          0           []

 arrival_lon (InputLayer)       [(None, 1)]          0           []

 departure_delay (InputLayer)   [(None, 1)]          0           []

 concatenate_32 (Concatenate)   (None, 3)            0           ['departure_lon[0][0]',
                                                                  'arrival_lon[0][0]',
                                                                  'departure_delay[0][0]']

 dense_layer (Dense)            (None, 1)            4           ['concatenate_32[0][0]']

==================================================================================================
Total params: 4
Trainable params: 4
Non-trainable params: 0
__________________________________________________________________________________________________
```



Represent longitude and delay in buckets



```
departure_lon: [-156.0, -155.0, -154.0, -153.0, -152.0, -151.0, -150.0, -149.0, -148.0, -147.0, -146.0, -145.0, -144.0, -143.0, -142.0, -141.0, -140.0, -139.0, -138.0, -137.0, -136.0, -135.0, -13
4.0, -133.0, -132.0, -131.0, -130.0, -129.0, -128.0, -127.0, -126.0, -125.0, -124.0, -123.0, -122.0, -121.0, -120.0, -119.0, -118.0, -117.0, -116.0, -115.0, -114.0, -113.0, -112.0, -111.0, -110.
0, -109.0, -108.0, -107.0, -106.0, -105.0, -104.0, -103.0, -102.0, -101.0, -100.0, -99.0, -98.0, -97.0, -96.0, -95.0, -94.0, -93.0, -92.0, -91.0, -90.0, -89.0, -88.0, -87.0, -86.0, -85.0, -84.0,
-83.0, -82.0, -81.0, -80.0, -79.0, -78.0, -77.0, -76.0, -75.0, -74.0, -73.0, -72.0, -71.0, -70.0, -69.0, -68.0, -67.0, -66.0, -65.0]
arrival_lon: [-158.0, -157.0, -156.0, -155.0, -154.0, -153.0, -152.0, -151.0, -150.0, -149.0, -148.0, -147.0, -146.0, -145.0, -144.0, -143.0, -142.0, -141.0, -140.0, -139.0, -138.0, -137.0, -136.
0, -135.0, -134.0, -133.0, -132.0, -131.0, -130.0, -129.0, -128.0, -127.0, -126.0, -125.0, -124.0, -123.0, -122.0, -121.0, -120.0, -119.0, -118.0, -117.0, -116.0, -115.0, -114.0, -113.0, -112.0,
-111.0, -110.0, -109.0, -108.0, -107.0, -106.0, -105.0, -104.0, -103.0, -102.0, -101.0, -100.0, -99.0, -98.0, -97.0, -96.0, -95.0, -94.0, -93.0, -92.0, -91.0, -90.0, -89.0, -88.0, -87.0, -86.0, -
85.0, -84.0, -83.0, -82.0, -81.0, -80.0, -79.0, -78.0, -77.0, -76.0, -75.0, -74.0, -73.0, -72.0, -71.0, -70.0, -69.0, -68.0, -67.0]
departure_delay: [-14.0, -13.0, -12.0, -11.0, -10.0, -9.0, -8.0, -7.0, -6.0, -5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 1
6.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.
0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 61.0, 62.0, 63.0, 64.0, 65.0, 66.0, 67.0, 68.0, 69.0, 70.0, 71.0, 72.0, 73.0, 74.0, 75.0, 76.0, 77.0, 78.0, 79.0, 80.0,
81.0, 82.0, 83.0, 84.0, 85.0, 86.0, 87.0, 88.0, 89.0, 90.0, 91.0, 92.0, 93.0, 94.0, 95.0, 96.0, 97.0, 98.0, 99.0, 100.0, 101.0, 102.0, 103.0, 104.0, 105.0, 106.0, 107.0, 108.0, 109.0, 110.0, 111.
0, 112.0, 113.0, 114.0, 115.0, 116.0, 117.0, 118.0, 119.0, 120.0, 121.0, 122.0, 123.0, 124.0, 125.0, 126.0, 127.0, 128.0, 129.0, 130.0, 131.0, 132.0, 133.0, 134.0, 135.0, 136.0, 137.0, 138.0, 13
9.0, 140.0, 141.0, 142.0, 143.0, 144.0, 145.0, 146.0, 147.0, 148.0, 149.0, 150.0, 151.0, 152.0, 153.0, 154.0, 155.0, 156.0, 157.0, 158.0, 159.0, 160.0, 161.0, 162.0, 163.0, 164.0, 165.0, 166.0, 1
67.0, 168.0, 169.0, 170.0, 171.0, 172.0, 173.0, 174.0, 175.0, 176.0, 177.0, 178.0, 179.0, 180.0, 181.0, 182.0, 183.0, 184.0, 185.0, 186.0, 187.0, 188.0, 189.0, 190.0, 191.0, 192.0, 193.0, 194.0,
195.0, 196.0, 197.0, 198.0, 199.0, 200.0, 201.0, 202.0, 203.0, 204.0, 205.0, 206.0, 207.0, 208.0, 209.0, 210.0, 211.0, 212.0, 213.0, 214.0, 215.0, 216.0, 217.0, 218.0, 219.0, 220.0, 221.0, 222.0,
223.0, 224.0, 225.0, 226.0, 227.0, 228.0, 229.0, 230.0, 231.0, 232.0, 233.0, 234.0, 235.0, 236.0, 237.0, 238.0, 239.0, 240.0, 241.0, 242.0, 243.0, 244.0, 245.0, 246.0, 247.0, 248.0, 249.0, 250.0,
251.0, 252.0, 253.0, 254.0, 255.0, 256.0, 257.0, 258.0, 259.0, 260.0, 261.0, 262.0, 263.0, 264.0, 265.0, 266.0, 267.0, 268.0, 269.0, 270.0, 271.0, 272.0, 273.0, 274.0, 275.0, 276.0, 277.0, 278.0,
279.0, 280.0, 281.0, 282.0, 283.0, 284.0, 285.0, 286.0, 287.0, 288.0, 289.0, 290.0, 291.0, 292.0, 293.0, 294.0, 295.0, 296.0, 297.0, 298.0, 299.0, 300.0, 301.0, 302.0, 303.0, 304.0, 305.0, 306.0,
307.0, 308.0, 309.0, 310.0, 311.0, 312.0, 313.0, 314.0, 315.0, 316.0, 317.0, 318.0, 319.0, 320.0, 321.0, 322.0, 323.0, 324.0, 325.0, 326.0, 327.0, 328.0, 329.0, 330.0, 331.0, 332.0, 333.0, 334.0,
335.0, 336.0, 337.0, 338.0, 339.0, 340.0, 341.0, 342.0, 343.0, 344.0, 345.0, 346.0, 347.0, 348.0, 349.0, 350.0, 351.0, 352.0, 353.0, 354.0, 355.0, 356.0, 357.0, 358.0, 359.0, 360.0, 361.0, 362.0,
363.0, 364.0, 365.0, 366.0, 367.0, 368.0, 369.0, 370.0, 371.0, 372.0, 373.0, 374.0, 375.0, 376.0, 377.0, 378.0, 379.0, 380.0, 381.0, 382.0, 383.0, 384.0, 385.0, 386.0, 387.0, 388.0, 389.0, 390.0,
391.0, 392.0, 393.0, 394.0, 395.0, 396.0, 397.0, 398.0, 399.0, 400.0, 401.0, 402.0, 403.0, 404.0, 405.0, 406.0, 407.0, 408.0, 409.0, 410.0, 411.0, 412.0, 413.0, 414.0, 415.0, 416.0, 417.0, 418.0,
419.0, 420.0, 421.0, 422.0, 423.0, 424.0, 425.0, 426.0, 427.0, 428.0, 429.0, 430.0, 431.0, 432.0, 433.0, 434.0, 435.0, 436.0, 437.0, 438.0, 439.0, 440.0, 441.0, 442.0, 443.0, 444.0, 445.0, 446.0,
447.0, 448.0, 449.0, 450.0, 451.0, 452.0, 453.0, 454.0, 455.0, 456.0, 457.0, 458.0, 459.0, 460.0, 461.0, 462.0, 463.0, 464.0, 465.0, 466.0, 467.0, 468.0, 469.0, 470.0, 471.0, 472.0, 473.0, 474.0,
475.0, 476.0, 477.0, 478.0, 479.0, 480.0, 481.0, 482.0, 483.0, 484.0, 485.0, 486.0, 487.0, 488.0, 489.0, 490.0, 491.0, 492.0, 493.0, 494.0, 495.0, 496.0, 497.0, 498.0, 499.0, 500.0, 501.0, 502.0,
503.0, 504.0, 505.0, 506.0, 507.0, 508.0, 509.0, 510.0, 511.0, 512.0, 513.0, 514.0, 515.0, 516.0, 517.0, 518.0, 519.0, 520.0, 521.0, 522.0, 523.0, 524.0, 525.0, 526.0, 527.0, 528.0, 529.0, 530.0,
531.0, 532.0, 533.0, 534.0, 535.0, 536.0, 537.0, 538.0, 539.0, 540.0, 541.0, 542.0, 543.0, 544.0, 545.0, 546.0, 547.0, 548.0, 549.0, 550.0, 551.0, 552.0, 553.0, 554.0, 555.0, 556.0, 557.0, 558.0,
559.0, 560.0, 561.0, 562.0, 563.0, 564.0, 565.0, 566.0, 567.0, 568.0, 569.0, 570.0, 571.0, 572.0, 573.0, 574.0, 575.0, 576.0, 577.0, 578.0, 579.0, 580.0, 581.0, 582.0, 583.0, 584.0, 585.0, 586.0,
587.0, 588.0, 589.0, 590.0, 591.0, 592.0, 593.0, 594.0, 595.0, 596.0, 597.0, 598.0, 599.0, 600.0, 601.0, 602.0, 603.0, 604.0, 605.0, 606.0, 607.0, 608.0, 609.0, 610.0, 611.0, 612.0, 613.0, 614.0,
615.0, 616.0, 617.0, 618.0, 619.0, 620.0, 621.0, 622.0, 623.0, 624.0, 625.0, 626.0, 627.0, 628.0, 629.0, 630.0, 631.0, 632.0, 633.0, 634.0, 635.0, 636.0, 637.0, 638.0, 639.0, 640.0, 641.0, 642.0,
643.0, 644.0, 645.0, 646.0, 647.0, 648.0, 649.0, 650.0, 651.0, 652.0, 653.0, 654.0, 655.0, 656.0, 657.0, 658.0, 659.0, 660.0, 661.0, 662.0, 663.0, 664.0, 665.0, 666.0, 667.0, 668.0, 669.0, 670.0,
671.0, 672.0, 673.0, 674.0, 675.0, 676.0, 677.0, 678.0, 679.0, 680.0, 681.0, 682.0, 683.0, 684.0, 685.0, 686.0, 687.0, 688.0, 689.0, 690.0, 691.0, 692.0, 693.0, 694.0, 695.0, 696.0, 697.0, 698.0,
699.0, 700.0, 701.0, 702.0, 703.0, 704.0, 705.0, 706.0, 707.0, 708.0, 709.0, 710.0, 711.0, 712.0, 713.0, 714.0, 715.0, 716.0, 717.0, 718.0, 719.0, 720.0, 721.0, 722.0, 723.0, 724.0, 725.0, 726.0,
727.0, 728.0, 729.0, 730.0, 731.0, 732.0, 733.0, 734.0, 735.0, 736.0, 737.0, 738.0, 739.0, 740.0, 741.0, 742.0, 743.0, 744.0, 745.0, 746.0, 747.0, 748.0, 749.0, 750.0, 751.0, 752.0, 753.0]
```

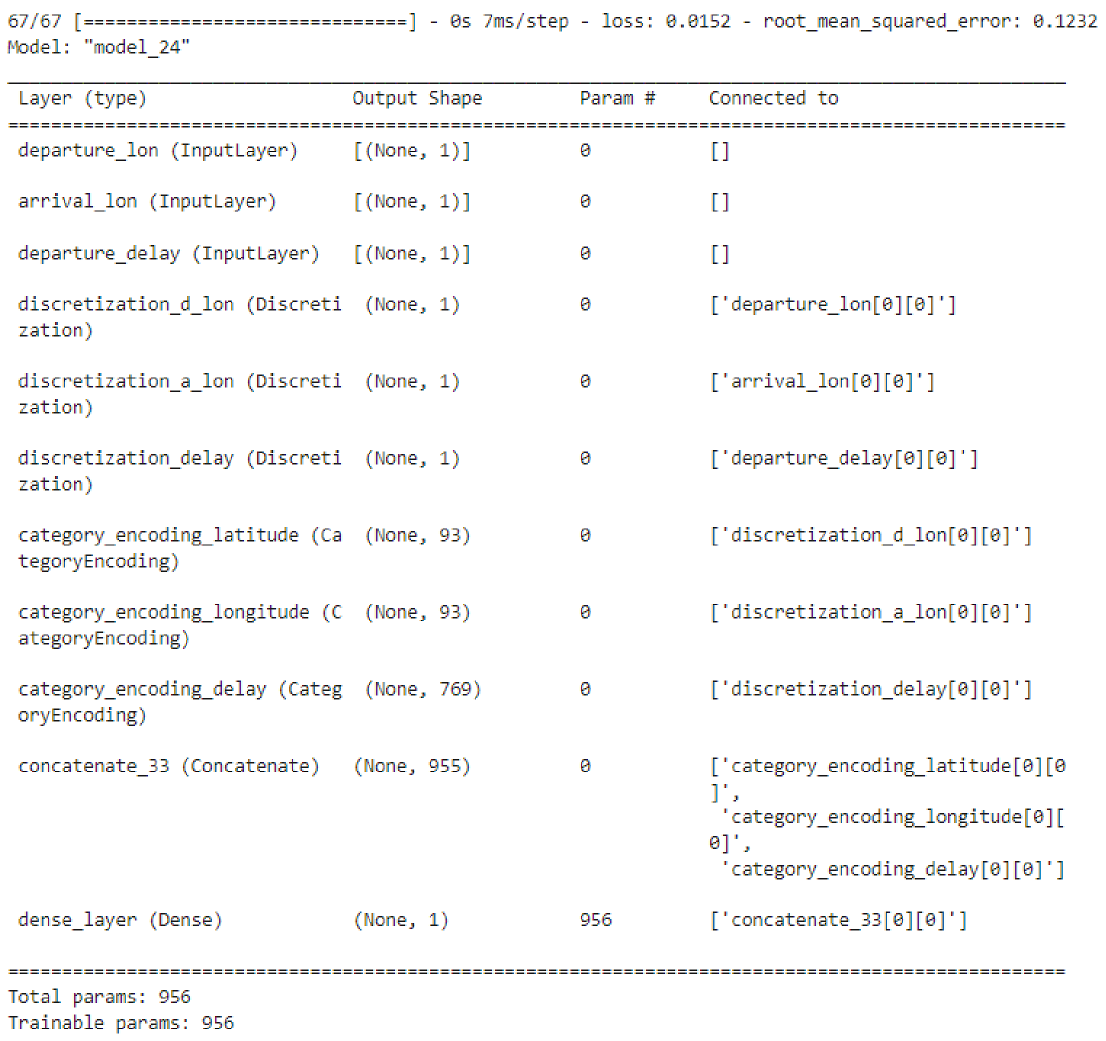


```
[74]: # The following variables are the hyperparameters.
      learning_rate = 0.04
      epochs = 35

      # Build the model.
      my_model = create_model(inputs, outputs, learning_rate)


      # Train the model on the training set.
      epochs, rmse = train_model(my_model, train_df,
                                 epochs,
                                 batch_size,
                                 label_name)

      # Print out the model summary.
      my_model.summary(expand_nested=True)

      plot_the_loss_curve(epochs, rmse)
```

```
Epoch 1/35
67/67 [==============================] - 1s 7ms/step - loss: 2321.8091 - root_mean_squared_error: 48.1852
Epoch 2/35
67/67 [==============================] - 0s 7ms/step - loss: 1501.3794 - root_mean_squared_error: 38.7476
Epoch 3/35
67/67 [==============================] - 0s 7ms/step - loss: 873.3067 - root_mean_squared_error: 29.5518
Epoch 4/35
67/67 [==============================] - 0s 7ms/step - loss: 422.0848 - root_mean_squared_error: 20.5447
Epoch 5/35
67/67 [==============================] - 0s 7ms/step - loss: 142.7057 - root_mean_squared_error: 11.9460
Epoch 6/35
67/67 [==============================] - 0s 7ms/step - loss: 23.8083 - root_mean_squared_error: 4.8794
Epoch 7/35
67/67 [==============================] - 0s 7ms/step - loss: 5.4915 - root_mean_squared_error: 2.3434
Epoch 8/35
67/67 [==============================] - 0s 7ms/step - loss: 2.8920 - root_mean_squared_error: 1.7006
Epoch 9/35
67/67 [==============================] - 0s 7ms/step - loss: 1.7138 - root_mean_squared_error: 1.3091
Epoch 10/35
67/67 [==============================] - 0s 7ms/step - loss: 1.1167 - root_mean_squared_error: 1.0567
Epoch 11/35
67/67 [==============================] - 0s 7ms/step - loss: 0.7696 - root_mean_squared_error: 0.8773
```

We can see that loss is reduced to 0.0512 and RMSE is reduced to 0.1232.

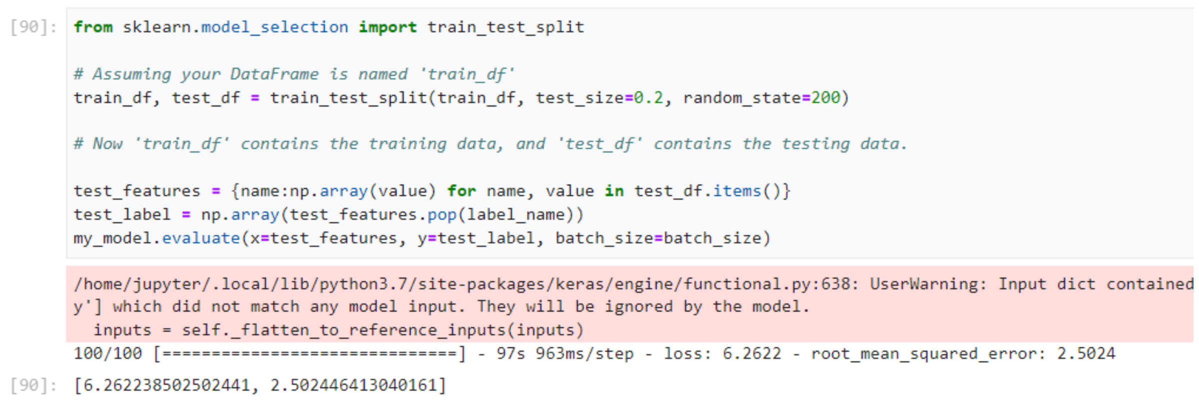
In comparison to the original baseline model which had an RMSE of ~54, this is a significant improvement. Moreover, with the limited information and an SD of ~42, an RMSE of ~0.12 can be considered rather accurate.



```
67/67 [==============================] - 0s 7ms/step - loss: 0.0152 - root_mean_squared_error: 0.1232
Model: "model_24"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 departure_lon (InputLayer)     [(None, 1)]          0           []

 arrival_lon (InputLayer)       [(None, 1)]          0           []

 departure_delay (InputLayer)   [(None, 1)]          0           []

 discretization_d_lon (Discreti  (None, 1)           0           ['departure_lon[0][0]']
 zation)

 discretization_a_lon (Discreti  (None, 1)           0           ['arrival_lon[0][0]']
 zation)

 discretization_delay (Discreti  (None, 1)           0           ['departure_delay[0][0]']
 zation)

 category_encoding_latitude (Ca  (None, 93)          0           ['discretization_d_lon[0][0]']
 tegoryEncoding)

 category_encoding_longitude (C  (None, 93)          0           ['discretization_a_lon[0][0]']
 ategoryEncoding)

 category_encoding_delay (Categ  (None, 769)         0           ['discretization_delay[0][0]']
 oryEncoding)

 concatenate_33 (Concatenate)   (None, 955)          0           ['category_encoding_latitude[0][0
                                                                 ]',
                                                                  'category_encoding_longitude[0][
                                                                 0]',
                                                                  'category_encoding_delay[0][0]']

 dense_layer (Dense)            (None, 1)            956         ['concatenate_33[0][0]']

==================================================================================================
Total params: 956
Trainable params: 956
```

Splitting data set to test our model

We can see that we have a data loss of 6 and RMSE of 2.5 which is pretty accurate.



```
[90]: from sklearn.model_selection import train_test_split

      # Assuming your DataFrame is named 'train_df'
      train_df, test_df = train_test_split(train_df, test_size=0.2, random_state=200)

      # Now 'train_df' contains the training data, and 'test_df' contains the testing data.

      test_features = {name:np.array(value) for name, value in test_df.items()}
      test_label = np.array(test_features.pop(label_name))
      my_model.evaluate(x=test_features, y=test_label, batch_size=batch_size)

      /home/jupyter/.local/lib/python3.7/site-packages/keras/engine/functional.py:638: UserWarning: Input dict contained
      y'] which did not match any model input. They will be ignored by the model.
        inputs = self._flatten_to_reference_inputs(inputs)
      100/100 [==============================] - 97s 963ms/step - loss: 6.2622 - root_mean_squared_error: 2.5024
[90]: [6.262238502502441, 2.502446413040161]
```