Key Steps in Machine Learning Model Development - Chicago Taxi Trips Use Case

Business goal and machine learning solution

In this section we provide the business goal for this Demo (which is is to be able to have reliable predictions about total trip duration so that these predictions can be used, in a future step, on apps used by taxi clients), as well as the machine learning use case (to provide a complete workflow in GCP, for developing and deploying predictive models based on Deep Learning Regression models to predict the total taxi travel durations in Chicago city) and how the solution is expected to address the business goal (this machine learning solution is expected to address the business goal because of superior predictive of deep learning models compared with other predictive machine learning family models).

Considering the Chicago Trips dataset, we decided on an approach of building a machine learning solution with deep learning neural networks to predict how much time the taxi trip will take, based on the historical data and machine learning prediction.

In this section we will perform a description of the identified business needs being addressed in this demo, and how the proposed machine learning solution will address this business need translated into a business goal.

As it will be depicted in this document, one key business need was identified and approached in this use case: the necessity to have good, accurate and responsive predictions for the total duration of the taxi trips made by any given taxi company in Chicago city.

To have such predictions allows a taxi company can offer this information to its customers in a manageable time, for instance, it can offer these information/predictions to a customer demanding its services/ demanding a given taxi trip, by means of a cell phone app service.

Henceforth, the identified business goal for this Demo is to be able to provide reliable predictions about total trip duration so that these predictions can be used, in a future step, on apps used by taxi clients.

In this sense, a machine learning solution (final trained model) can be served in order to provide online predictions to these apps in order to manage/increase customer satisfaction, keeping them informed of the expected trip duration at the moment the trip orders are being made.

These predictions can also be used to further improve service levels and strategic planning procedures where some taxi travel routes can be improved by making more taxis available for these routes, in order to reduce some journeys with very long travel times.

Hence, the Machine Learning use case is defined as to present a complete machine learning workflow, ranging from data exploration to the machine learning model deployment in order to provide these taxi trips time length predictions. The whole workflow proposed in the use case will be developed using exclusively Google Cloud services.

The proposed Machine Learning model to keep up this business need was a Deep Learning Regression model, deployed in Google Cloud Model Repository to an endpoint so that the deployed model can receive requests and provide forecasts for the total time length (in minutes) of any given trip. This machine learning solution is expected to address the business goal because of superior predictive of deep learning models compared with other predictive machine learning family models, given available large datasets (which is the case in this demo) and proper training time efforts.

We have decided to provide a Deep Learning model due to the size of this dataset and also because of the requisites for this use case development.

As described above, the goal of this development is to present an initial workflow, made entirely of Google Cloud services, in order to offer a deployed a machine model aimed to keep up with this business need (which is to predict the total trip time of the taxi trips in Chicago -measured in total trip minutes- , given observed values for a set of selected features).

For this purpose one used the chicago taxi trips dataset, in order to develop and present this complete workflow.

The adopted definition of done for the developments to be implemented below is a presentation of a complete workflow (ranging from data exploration to model deployment and test) in order to make an initial machine learning available for predictions of the aforementioned time durations of the taxi trips in Chicago city (regardless of the model performance).

The main objective of this use case is to illustrate how to implement a complete workflow for this purpose (or course suggestions for next steps in these developments and what can be done in order to improve initial results).

As it will pointed out in the developments below, this work had to handle a set of constraints (specially time and resource constraints) which by themselves shaped/ limited many of the decisions adopted along this use case (such as hardware/ infrastructure to handle computing procedures, the hyperparameters adopted in the model training sessions and so on). However, given these constraints, the goal of the Demo 1, which is to provide an implementation of the complete machine workflow in Google Cloud in order to make a ML solution available to provide the aforementioned predictions, was achieved.

Data exploration

Regarding data exploration we present the types of data exploration implemented (missing data, checking data types, correlation analysis and feature statistical empirical distribution analysis). We then present the main decisions made based on the exploratory analysis and how it influenced the models (mainly we have identified the target variable to be predicted, as well as discarding some variable because of redundant information - specifically in the case of the dropoff_location and pickup_location features -).

Description of the types of data exploration implemented and how they were performed

For the purposes of data analysis, we have used a specific python Notebook, in Vertex AI/ Workbench in order to implement it.

The goals of the exploratory data analysis on the chicago-taxi-trips dataset were:

- Identify dataset size and develop a strategy to perform exploratory data analysis on it based on this size and which tools are to be used on this exploratory data analysis.

- To identify data types on this dataset and possible necessities to change them for a given subset of columns in the original, raw dataset.

- To identify possible target variables for the predictive model to be developed on this initiative, that is, variables to be predicted by the Deep Learning model to be built. Also choose a final target variable to be predicted in the original dataset.

- Identify columns that can be immediately discarded for the purposes of training and delivering a predictive model for a chosen target variable.

- Identify empirical distributions of the variables on the dataset and if any transformations on the initial variables are to be implemented.

- Correlation analysis for the variables in the dataset and which variables are to be kept considering the correlation patterns identified in the data .

- Check the existence of missing values for each of the variables contained in the dataset and which action to be taken regarding these missing values for each initial variable.

- Check the possibility of creating new features that are meaningful for the model to be trained, from the initial variables in the chicago-taxi-trips public dataset.

- Check other type of transformations and scaling on the data are necessary for the Deep Learning model to be developed.

Steps followed on the exploratory data analysis:

Initially, we have copied the public Chicago Taxi Trips dataset to the a new BigQuery schema,to be used in the ML Specialization initiative project. The name of the project created for this ML especialization is gcp-ml-especialization-2024, and the created schema in BigQuery to be used was spec_taxi_trips.

After copying this dataset to the new BigQuery schema we have checked the data types on the Big Query, and the initial fields on the dataset, both by running a SELECt statement as well as checking informations on the BigQuery preview function.

By running a SELECT COUNT(*) statement on BigQuery query editor we were able to check the number of rows in the dataset.

Due to the number of rows in the dataset (more than 102 millions of rows, 102.589.284 to be precise) we have decided to carry on the next step in exploratory data analysis with a Python notebook, using a sample from this dataset (for this sample we have decided to consider a sample formed with observations ranging from trip_start_timestamp 2013-01-01T00:00:00.000Z to 2015-01-01T00:00:00.000Z).

With this sample (composed of 23,033,713 rows and 23 columns) the exploratory analysis was carried out in this manner:

Visualizing this sample as a pandas dataframe we have identified one column that has no usefulness for the purpose of prediction, which is the unique_key field, which is a unique identifier of the taxi trips. This field was discarded from any further consideration.

We have identified potential candidate variables for target variables for a predictive model. We have identified such variables as being 'trip_seconds' and 'trip_total'. We have decided to keep 'trip_seconds' as a target variable for the ML model to be considered. Also, we have decided to make a transformation on this variable in order to make it more interpretative: we have decided to change the scale of this variable to minutes, in order to make it more interpretative. This was later accomplished in the feature engineering step, to be described later, in the feature engineering session of this document.

Due to the size of the dataset we have also discarded in this step the column, taxi_id, which was also discarded for any further consideration.

We checked the data types of the variables present in the chicago_taxi_trips public dataset. For the case of the trip_Seconds variable, we have decided to change its data type to float, as described later.

We checked the distribution of the features involved in the raw, original dataset.

Data types present in the original raw dataset

The existence of missing values on the original columns of the dataset.

We also checked the correlation patterns amongst pairs of features. Some strong correlations between specific pairs were detected. For these cases we have decided to discard one of the two strongly correlated variables.

Key findings in the data exploration step

- Correlation Patterns key findings:

As shown in the correlation matrix below, we have identified that variables fare and trip_total were strongly correlated. We have decided to discard fare and keep trip_toral as this last variable (trip_total).

We also have identified some relatively important correlations between the following pairs of variables:

- Tolls and trip_total, (Pearson correlation of magnitude 0.48),
- Tips and trip_miles (Pearson correlation of magnitude 0.42),
- Pickup_longitude and trip_miles (Pearson correlation of magnitude -0.47),
- dropoff_longitude and trip_miles (Pearson correlation of magnitude -0.44),
- Tolls and fare (Pearson correlation of magnitude 0.46),

Because Tolls have meaningful correlations with different variables it was aso immediately discarded from the dataset.

We have also identified a correlation of 0.41 between pickup_latitude and dropoff_latitude.

Taking into account this last correlation pattern and also the size of the initial dataset (and the limitations of the available hardware/ machine for the development of this use case), we have also decided to discard the dropoff_latitude and dropoff_longitude pair from further consideration to train a Deep Learning model for this Specialization (inn order to make predictions for the trip_minutes target variables)..

CODE SNIPET:

```
continuous_features = sample2_taxi[['trip_seconds', 'trip_total', 'trip_miles',
       'fare', 'tips', 'tolls',
       'extras', 'pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff_longitude']]

corr = continuous_features.corr()
corr.style.background_gradient()
```

Figure 5: Correlation matrix between features in sample dataset

	trip_seconds	trip_total	trip_miles	fare	tips	tolls	extras	pickup_latitude	pickup_longitude	dropoff_latitude	dropoff_longitude
trip_seconds	1.000000	0.064512	0.343613	0.057877	0.172658	0.004296	0.085710	0.077951	-0.192256	0.054367	-0.174943
trip_total	0.064512	1.000000	0.129496	0.998820	0.079628	0.477256	0.069948	0.026565	-0.072503	0.016417	-0.059548
trip_miles	0.343613	0.129496	1.000000	0.111960	0.420391	-0.005422	0.247544	0.178705	-0.470087	0.122613	-0.440947
fare	0.057877	0.998820	0.111960	1.000000	0.048730	0.462109	0.030530	0.022542	-0.058809	0.015041	-0.051695
tips	0.172658	0.079628	0.420391	0.048730	1.000000	-0.006361	0.188705	0.085060	-0.304383	0.047834	-0.225737
tolls	0.004296	0.477256	-0.005422	0.462109	-0.006361	1.000000	0.103134	0.001203	-0.001039	0.001705	-0.002383
extras	0.085710	0.069948	0.247544	0.030530	0.188705	0.103134	1.000000	0.068908	-0.217990	0.026624	-0.201769
pickup_latitude	0.077951	0.026565	0.178705	0.022542	0.085060	0.001203	0.068908	1.000000	-0.539796	0.412341	-0.169189
pickup_longitude	-0.192256	-0.072503	-0.470087	-0.058809	-0.304383	-0.001039	-0.217990	-0.539796	1.000000	-0.159729	0.178885
dropoff_latitude	0.054367	0.016417	0.122613	0.015041	0.047834	0.001705	0.026624	0.412341	-0.159729	1.000000	-0.485302
dropoff_longitude	-0.174943	-0.059548	-0.440947	-0.051695	-0.225737	-0.002383	-0.201769	-0.169189	0.178885	-0.485302	1.000000

- we have identified unnecessary features like unike_key (unique identifiers of each row)

- **Missing Value Key Findings:**

Using the sample data, with a python notebook, we were able to initially find that some variables had a large amount of missing data in the original dataset.

For this, we found out the following results:

Figure 6: Null values along sample dataset

```
unique_key                  0
taxi_id                     0
trip_start_timestamp        0
trip_end_timestamp         64
trip_seconds            39325
trip_miles                  3
pickup_census_tract       6671516
dropoff_census_tract      7046153
pickup_community_area     1231247
dropoff_community_area    1998855
fare                        172
tips                        172
tolls                     5706924
extras                      172
trip_total                  172
payment_type                0
company                  17232429
pickup_latitude           1229555
pickup_longitude          1229555
pickup_location           1229555
dropoff_latitude          1956612
dropoff_longitude         1956612
dropoff_location          1956612
dtype: int64
```

CODE SNIPET:

```
### Checking null values

sample2_taxi.isnull().sum()
```

After checking the existence of null values for some variables in the sample dataset, we have proceeded to check the null values in all dataset. For that one carried out Big Query Queries to get the exact total number of missing values in the whole dataset.

The code snippet below shows the exact amount of missing values in the dataset.

CODE SNIPET:

```
SELECT                                    count(*)                    FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`; -- 102.589.284



SELECT                                    count(*)                    FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`   WHERE   fare   IS
NULL; -- 2821

SELECT                                    count(*)                    FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`   WHERE   tips   IS
NULL; -- 2821

SELECT                                    count(*)                    FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`   WHERE   tolls   IS
NULL; -- 24.526.374!!!

SELECT                                    count(*)                    FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`  WHERE  extras  IS
NULL; -- -- 2821

SELECT                                    count(*)                    FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`  WHERE  trip_total
IS NULL; -- -- 2821

SELECT                                    count(*)                    FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`              WHERE
payment_type IS NULL; -- -- o

SELECT                                    count(*)                    FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`  WHERE  company  IS
NULL; -- -- 29.590.255!!!
```

```sql
SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE payment_type IS NULL; -- -- 0

SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE fare IS NULL; -- -- 2821

SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE pickup_latitude IS NULL; -- -- 7.932.764

SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE pickup_longitude IS NULL; -- -- 7.932.764

SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE dropoff_latitude IS NULL; -- -- 9.753.459

SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE dropoff_longitude IS NULL; -- -- 9.753.459

SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE distance IS NULL; -- -- 9.753.459

SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE trip_start_timestamp IS NULL; --0

SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE month_trip_end IS NULL; --0

SELECT count(*) FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips` WHERE pickup_community_area IS NULL; --7.946.662
```

```
SELECT                                    count(*)                    FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`              WHERE
dropoff_community_area IS NULL; --10.022.389

month_trip_end
```

Given the existence of missing values in the dataset we have decided to discard completely all rows in the dataset containing missing values for the selected features, as will be described in the feature engineering session.

We also proceeded to carry on additional exploratory data analysis in BIG QUERY, such as checking the different values for some of the categorical variables present on the original dataset. This is shown below, on the following code snippet.

CODE SNIPET - checking different unique values of some of the categorical variables in the dataset.

```
SELECT                        DISTINCT            company            FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`; -- 41 different
types of companies



SELECT                      DISTINCT          payment_type           FROM
`gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`; -- 11 different
occurences of payment types
```

Initial raw variables content key findings:

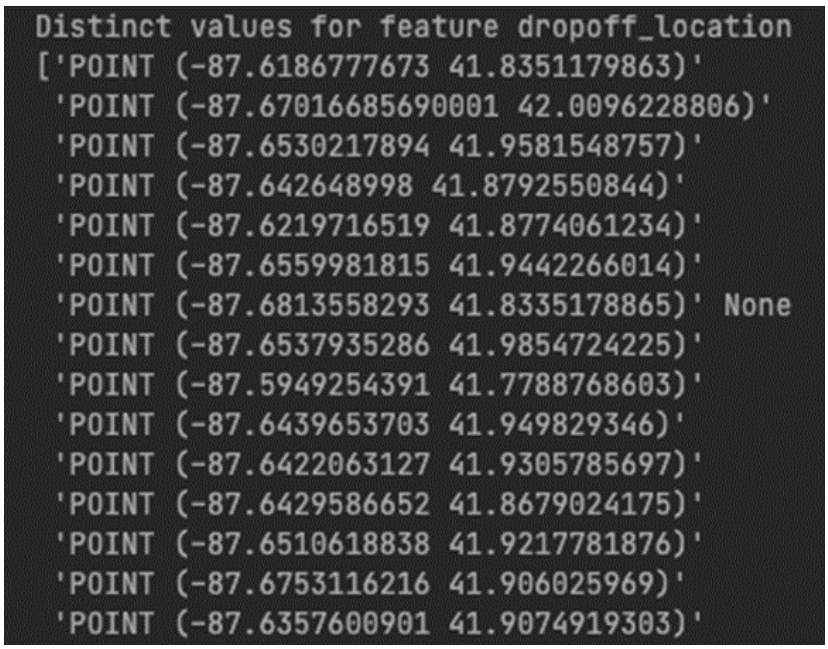
We found out that some original variables in the dataset contain information that somewhat were already present in another variables. This was the case for the following variables:

a) dropoff_location : containing geo location information about dropoff locations, as points of latitude/ longitude pairs. As the original dataset already contained

variables with these informations (variables dropoff_latitude and dropoff longitude) we decided to keep the later two and discard this variable. As it will be shown later, we will use these later variables - dropoff_latitude and dropoff longitude - , as well as variables pickup_latitude and pickup_longitude to buid a new engineered feature, called distance, which is the euclidean distance between pickup coordinates and dropoff coordinates).

Below is a snapshot of the feature dropoff_location containing a sample of its content:

Figure 7: Snapshot of sample dropoff_location feature content



In the same manner, we have discarded the pickup_location variable from any further consideration.

Below is a snapshot of the pickup_location sample content, containing POINT geolocation data.

Figure 8: snapshot of sample pickup_location sample content

```
Distinct values for feature pickup_location
['POINT (-87.6203346241 41.8571838585)'
 'POINT (-87.7696154528 41.7925923603)'
 'POINT (-87.6439653703 41.949829346)'
 'POINT (-87.6716536214 41.8786674201)'
 'POINT (-87.6997544065 41.9211259143)'
 'POINT (-87.6813558293 41.8335178865)'
 'POINT (-87.6542980837 41.9462945357)'
 'POINT (-87.6358909539 41.9292629902)'
 'POINT (-87.6525344838 41.8926581076)'
 'POINT (-87.6515625922 41.9363101308)'
 'POINT (-87.7049072355 41.9283913974)'
 'POINT (-87.6462934762 41.9290776551)'
 'POINT (-87.6564115308 41.9362371791)'
 'POINT (-87.6540926517 41.8716894745)'
 'POINT (-87.6460070664 41.9534000435)'
 'POINT (-87.6653376596 41.9066507664)'
```

Key findings related to trip time-related information:

Other key finding in the dataset, regarding original variables contents related to information relative to time and what statistical modelling/ data strategy were adopted regarding the content of these variables.

We have detected that only the variables trip_start_timestamp and trip_end_timestamp contain information relative to time (in UTC reference). Moreover, we can have repetitions of the same timestamp along different trips.

Because of that, we have decided to adopt a cross sectional analysis approach to data (instead of a time-series/ panel approach).

Because of this important decision, based on the analysis of the sample dataset, using Pandas dataframes in a python notebook in VErtex Workbench, we also identified the necessity of creating new features related to time, but with a cross sectional data approach: as it will be described in the Feature Engineering section, we will create new features to represent the

minute, hour, day, day of the week and month or each trip, by extracting these informations from these two original variables in the dataset (trip_start_timestamp and trip_end_timestamp). Once he have created these new features, from a feature engineering process, we will discard the initial two variables from any further consideration.

Below we have a snapshot of a sample of these two time-related variables:

CODE SNIPET:

```
%%bigquery sample2_taxi
SELECT *
FROM
  `bigquery-public-data.chicago_taxi_trips.taxi_trips`
 WHERE trip_start_timestamp BETWEEN '2013-01-01T00:00:00.000Z'
                 AND '2015-01-01T00:00:00.000Z';
```

```
sample2_taxi.head(3)
```

Figure 9: Sample content of trip_start_timestamp and trip_end_timestamp original variables in the dataset

	trip_start_timestamp	trip_end_timestamp	trip_seconds	trip_miles	pickup_census_tract	dropoff_census_tract
319fb34a956ed0b06101886deb51ce833dc...	2014-01-04 13:30:00+00:00	2014-01-04 13:45:00+00:00	445	2.3	<NA>	<NA>
0c09b7eb931aaa40b4157514f6466a12e2...	2014-12-15 20:30:00+00:00	2014-12-15 21:00:00+00:00	2380	20.5	<NA>	<NA>
7810af6cb50e01b15ca36acc3fb111193d3...	2014-12-19 19:30:00+00:00	2014-12-19 19:45:00+00:00	961	8.9	<NA>	<NA>

Key findings regarding distribution of the variables in the dataset:

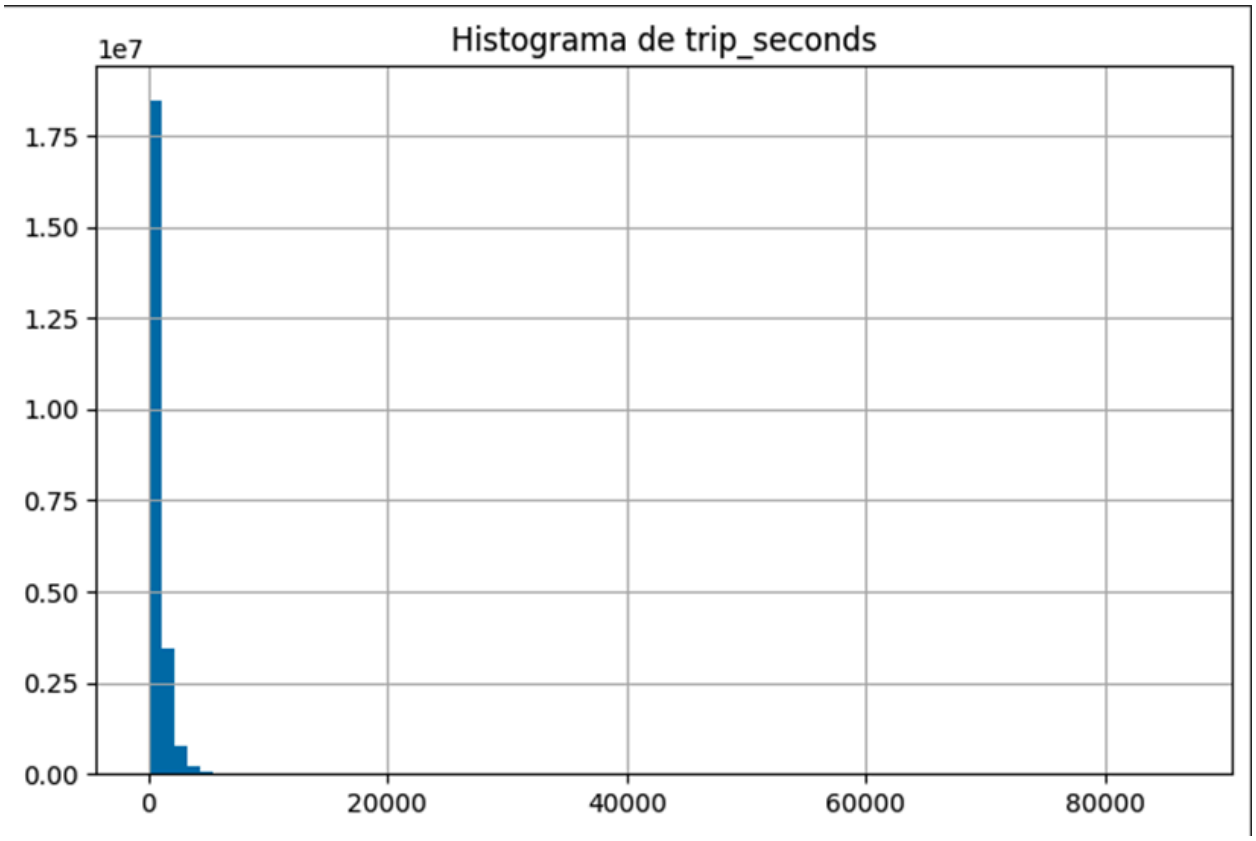
We have investigated the distribution of the variables on the original dataset.

For this purpose we have built histograms and bar charts (depending if the considered variable is continuous or discrete) to get some insight about the distribution of the data (using the sample dataset before mentioned).

We found out that most of the variables in the dataset have asymmetric distributions, given the Random simple sample from it.

To illustrate these cases we show below the sample histogram of the original target variable, trip_Seconds, before its transformation into trip_minutes. As it can be seen its sample distribution has a right-hand skew.

Figure 10 trip_seconds sample distribution:



.

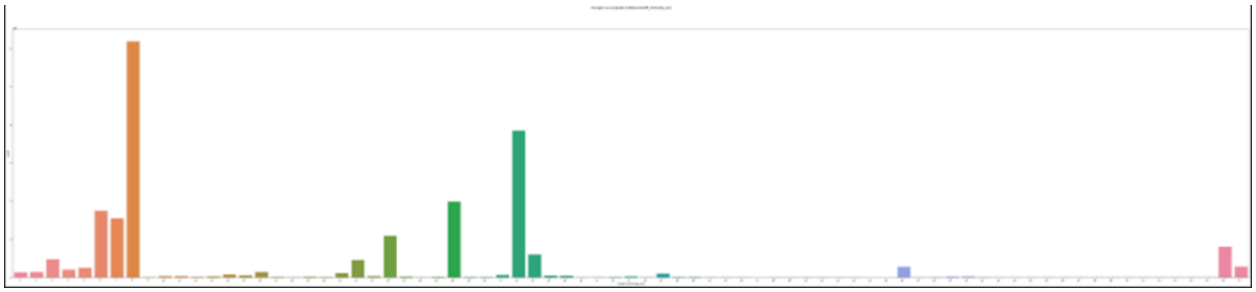
CODE SNIPET:

```
### Checking distribution of target variable trip_Seconds

plt.figure(figsize=(8,5))
plt.title("Histograma de trip_seconds")
hist = sample2_taxi['trip_seconds'].hist(bins=80)
ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha="right")
#plt.tight_layout()
plt.show()
```

In other cases, the distributions of some variables had several peaks (modes) like dropoff_community_area variable, as depicted below:

Figure 11: Barplot of dropoff_community_area

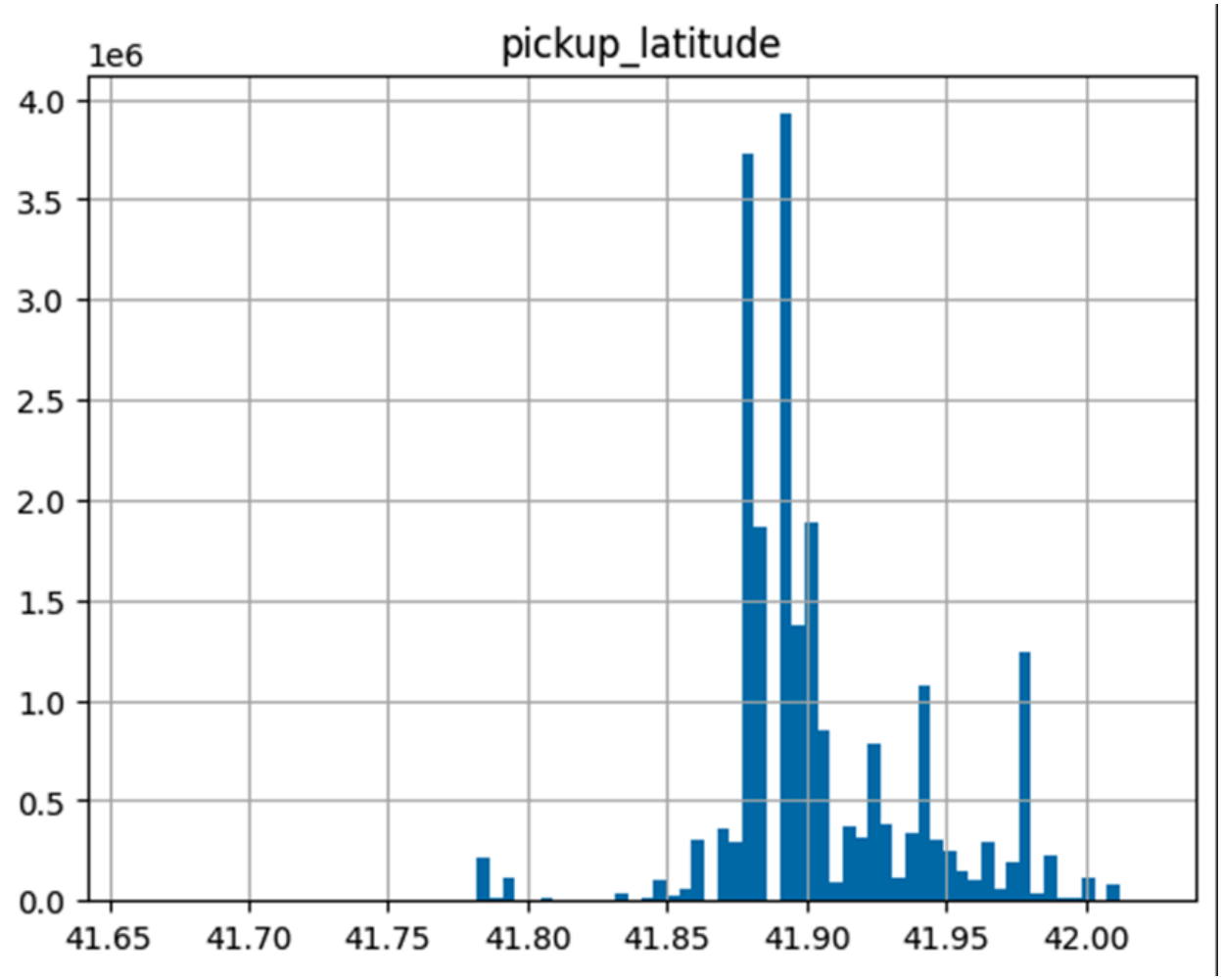


Finally some variables presented a somewhat symmetric distribution, like the variable pickup_latitude, as presented below.

CODE SNIPET:

```
plt.figure(figsize=(80,18))
plt.title("Contagem das categorias da feature dropoff_community_area", y=1.08)
ax = sns.countplot(x='dropoff_community_area',data=df1)
ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha="right")
plt.tight_layout()
```

Figure 12: pickup latitude sample distribution (histogram)



CODE_SNIPET:

```
sample2_taxi.hist(column='pickup_latitude', bins=80);
```

Although these are important findings, given that we will train a Deep Neural Network regressor for that target variable trip_minutes, it is not necessary to implement any data transformations in order to change the distributions of the features and the response variable.

For this reason and also for time limitations to implement this Specialization material, no data transformation was implemented in other to achieve this goal (to change data distribution).

We did implement, however, other type of data transformation, in order to speed up Deep Learning model training, as it will be depicted below.

The key findings in the data explorations steps were:

- Some variables had to had their data types changed. This was the case for the trip_seconds variable, that had data type integer and had it changed to float 64 type. This was due the necessity of later, in the feature engineer step, to divide its value by 60 in order to create a new feature, called trip_minutes (which is the duration of each trip in minutes, which is more interpretative than the duration in seconds).

Data Transformations

In order to speed up Deep Learning models training, we did implement a Min Max Scaler scaling process of the continuous variables. This was accomplished in the same SQL script implemented in BIGQUERY for feature engineering. This will be described in the next section.

Feature engineering

We describe in this section the types of feature engineering steps performed in the demo, using BigQuery.

The feature engineering steps were made of:

- **discarting null values for each feature, building new features from existing variables (such as creating MINUTE, HOUR and DAY features from initial timestamp features and, in the case of the original variable trip_seconds, we created the final target variable trip_minutes, by transforming the original variable trip_Seconds),**
- **Changing data types (such as in the case of "trip_seconds"),**
- **eliminating additional redundant variables were also performed.**
- **we made a final transformation (Min-MAx scaling) on continuous variables and**
- **performed a final feature selection (based on redundant information contained in some features, correlation patterns already identified and other reasons).**

Due to the dataset size we had to further reduce the number of features considered, getting to the final set of features: Hour_trip_start, Day_of_week_trip_start, Month_trip_start, Trip_minutes (the target variable for the model), Trip_miles, Pickup_latitude, Pickup_longitude and Distance. Totalling 7 features and one target variable.

Due to the size of the original dataset and complex nature of feature engineering, we have not used Google's feature store to feed data to the model. Instead, we have consumed the dataset directly from BigQuery to perform model training, validation and testing.

Due to the size of the original dataset, all feature engineering steps implemented in this use case were carried out in a single BIG QUERY script (named feature_engineering_final.sql , available in the project`s repository).

We have decided to implement all feature engineering steps in BIg QUery mainly because of performance and processing time.

The feature engineering script implements several successive steps in this regard, following the specified order:

1- In the first step of the feature engineering process, we select the desired fields from the dataset (discarding the already above mentioned variables), and we also create some of new features, by extracting the trip minutes, hours, days, days of the week and months from the timestamp variables originally present in the dataset (trip_start_timestamp and trip_end_timestamp).

2- In the second step we discard the null values for each selected variable in the previous step,

3 - Next we implement the Label encoding of the values of the categorical variables present in the dataset,

4 - We created a new feature, called distance, which is the Euclidean distance between pickup coordinates and dropoff coordinates,

5 - Finally we implement the scaling of the continuous features, implementing a Min-Max scaling of them, in order to speed up training of the Deep Learning model, to be accomplished afterwards.

After these steps, we stored the resulting dataset in the following Big Query table: `**gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_dataset1**`.

As it will be described in the section 3.1.3.6 Further elimination of variables in the dataset for model training, further feature eliminations were carried out in order to get to final train, validation and test datasets. As it can be verified in this section, the main reason for these further eliminations is to couple with available resources (especially due to the dataset size, we have decided to keep the most of the observations for each final considered feature, even if this would lead to further feature elimination).

Below we present the code snippet of the sql script responsible to carry on all these feature engineering steps in Google Big Query:

CODE SNIPET:

```
------------------ FINAL FEATURE ENGINEERING AND FEATURE SELECTION SCRIPT
-----

-- Feature engineering, dropping variables with many NULL values

WITH temp1 AS (

    SELECT

        --taxi_id,

        --trip_start_timestamp,

        --trip_end_timestamp,

        EXTRACT(MINUTE FROM trip_start_timestamp AT TIME ZONE "UTC") as
minute_trip_start,

        EXTRACT(HOUR FROM trip_start_timestamp AT TIME ZONE "UTC") as
hour_trip_start,

        EXTRACT(DAY FROM trip_start_timestamp AT TIME ZONE "UTC") as
day_trip_start,

        EXTRACT(DAYOFWEEK FROM trip_start_timestamp AT TIME ZONE "UTC") as
day_of_week_trip_start,

        EXTRACT(MONTH FROM trip_start_timestamp AT TIME ZONE "UTC") as
month_trip_start,
```

```
        --EXTRACT(YEAR FROM trip_start_timestamp AT TIME ZONE "UTC") as
year_trip_start,




        EXTRACT(MINUTE FROM trip_end_timestamp AT TIME ZONE "UTC") as
minute_trip_end,

        EXTRACT(HOUR FROM trip_end_timestamp AT TIME ZONE "UTC") as
hour_trip_end,

        EXTRACT(DAY FROM trip_end_timestamp AT TIME ZONE "UTC") as
day_trip_end,

        EXTRACT(DAYOFWEEK FROM trip_end_timestamp AT TIME ZONE "UTC") as
day_of_week_trip_end,

        EXTRACT(MONTH FROM trip_end_timestamp AT TIME ZONE "UTC") as
month_trip_end,

       -- EXTRACT(YEAR FROM trip_end_timestamp AT TIME ZONE "UTC") as
year_trip_end,



        CAST(trip_seconds AS float64)/60  AS trip_minutes,

        trip_miles,

        --pickup_census_tract,

        --dropoff_census_tract,

        pickup_community_area,

        dropoff_community_area,

        fare,

        tips,

        --tolls,
```

```
        extras,

        trip_total,

        payment_type,

        company,

        ROUND(pickup_latitude,3) as pickup_latitude,

        ROUND(pickup_longitude,3) as pickup_longitude,

        --pickup_location,

        ROUND(dropoff_latitude,3) as dropoff_latitude,

        ROUND(dropoff_longitude,3) as dropoff_longitude,

        --dropoff_location


ST_DISTANCE(ST_GEOGPOINT(pickup_longitude,pickup_latitude),ST_GEOGPOINT(dr
opoff_longitude,dropoff_latitude)) as distance



        FROM `gcp-ml-especialization-2024.spec_taxi_trips.taxi_trips`

        --WHERE trip_start_timestamp ='2013-01-01T00:00:00.000Z'



)

-- Eliminating NULL VALUES in all fields

, temp2 as (

    SELECT * FROM temp1

    WHERE

    minute_trip_start IS NOT NULL AND

    hour_trip_start IS NOT NULL AND
```

```
 day_trip_start IS NOT NULL AND

 day_of_week_trip_start IS NOT NULL AND

 month_trip_start IS NOT NULL AND

 minute_trip_end IS NOT NULL AND

 hour_trip_end IS NOT NULL AND

 day_trip_end IS NOT NULL AND

 day_of_week_trip_end IS NOT NULL AND

 month_trip_end IS NOT NULL AND


 trip_minutes IS NOT NULL AND

 trip_miles IS NOT NULL AND

 pickup_community_area IS NOT NULL AND

 dropoff_community_area IS NOT NULL AND

fare IS NOT NULL AND

--tolls IS NOT NULL AND

extras IS NOT NULL AND

trip_total IS NOT NULL AND

payment_type IS NOT NULL AND

company IS NOT NULL AND

pickup_latitude IS NOT NULL AND

pickup_longitude IS NOT NULL AND

dropoff_latitude IS NOT NULL AND

dropoff_longitude IS NOT NULL AND
```

```
   distance IS NOT NULL


)
-- Label encoding of company categorical variable
, temp3 as (


    SELECT
    company,
    DENSE_RANK() OVER (ORDER BY company ASC) AS company_num
    FROM
    (


        SELECT DISTINCT company FROM temp2
    )


)


, temp4 as (
    SELECT * FROM temp2
    LEFT JOIN temp3 USING (company)
)


-- Label encoding of payment_type categorical variable
```

```
, temp5 as (


    SELECT
    payment_type,
    DENSE_RANK() OVER (ORDER BY payment_type ASC) AS payment_type_num
    FROM
    (


        SELECT DISTINCT payment_type FROM temp4
    )


)


, temp6 as (
    SELECT * FROM temp4
    LEFT JOIN temp5 USING (payment_type)
)


-------
-- Min Max Scaler of continuous variables to speed up trainning, to
Improve Data Stability and Performance in Deep Learning
, temp7 as (
SELECT
        --taxi_id,
```

```
      --trip_start_timestamp,

      --trip_end_timestamp,

      payment_type,

      company,

      minute_trip_start,

      hour_trip_start,

      day_trip_start,

      day_of_week_trip_start,

      month_trip_start,

      --year_trip_start,




      minute_trip_end,

      hour_trip_end,

      day_trip_end,

      day_of_week_trip_end,

      month_trip_end,

     -- year_trip_end,


      ML.MIN_MAX_SCALER(trip_minutes) OVER() as trip_minutes,

      ML.MIN_MAX_SCALER(trip_miles) OVER() as trip_miles,

      --pickup_census_tract,

      --dropoff_census_tract,
```

```
        pickup_community_area,

        dropoff_community_area,

        ML.MIN_MAX_SCALER(fare) OVER() as fare,

        ML.MIN_MAX_SCALER(tips) OVER() as tips,

        --tolls,

        ML.MIN_MAX_SCALER(extras) OVER() as extras,

        ML.MIN_MAX_SCALER(trip_total) OVER() as trip_total,

        ML.MIN_MAX_SCALER(pickup_latitude) OVER() as pickup_latitude,

        ML.MIN_MAX_SCALER(pickup_longitude) OVER() as pickup_longitude,

        --pickup_location,

        ML.MIN_MAX_SCALER(dropoff_latitude) OVER() as dropoff_latitude,

        ML.MIN_MAX_SCALER(dropoff_longitude) OVER() as dropoff_longitude,

        --dropoff_location,

        ML.MIN_MAX_SCALER(distance) OVER() as distance,

        company_num,

        payment_type_num



        FROM temp6


)


-------

, final1 as (
```

```
    SELECT * FROM temp7
)
select * from final1
```

Further elimination of variables in the dataset for model training

Due to the still very large size of these datasets, and the available infrastructure available for the development of this demo, it was still necessary to further reduce the dataset size, which forced to further reducing the number of features considered for Deep Learning Model training (following the data strategy of keeping most of the rows for each feature, even that this leads to the sacrifice of reducing even further the number of variables for model training).

In this sense we further discarded the following variables for model training, validation and test:

- Day_trip_start (this feature was discarded because we already had the day_of_the_week_trip_start variable in the dataset),
- Hour_trip_end,
- Day_trip_end,
- Day_of_the_week_trip_end,
- Fare,
- Dropoff_latitude,
- Dropoff_longitude
- Company_num

With these choices, we ended up with final datasets composed of the following final features:

- Hour_trip_start,

- Day_of_week_trip_start,
- Month_trip_start,
- Trip_minutes (the target variable for the regression, Deep Learning modelo to be trained),
- Trip_miles,
- Pickup_latitude,
- Pickup_longitude,
- Distance.

Totalling 7 features and one target variable.

Data Preprocessing and the data pipeline (including final decisions regarding data strategies to be adopted on this use case)

All preprocessing steps implemented in this demo corresponded to the feature engineering steps described previously. The data pipeline adopted in this demo is composed of a Feature engineering phase (made entirely in Big Query), the saving resulting of this data in final tables (train validation and test tables in BigQuery, as already mentioned). Then the data written in these tables are read as Pandas dataframes so that models can be trained/ validated and tested.

As depicted in the previous section, all data preprocessing was carried out using Big Query scripts . All detailed information about all data preprocessing is described in this previous section. The final tables used in model training, validation and test were made available by the Big Query functionality/API that is callable from within the Jupyter notebooks, to read Big Query tables as Pandas dataframes from within Vertex Workbench for model training validation and testing.

For the deployed model, the model endpoint is able to receive requests in JSON formats, in .csv files sitting in Cloud Storage or reading from Big Query tables using Big Query API, both for batch predictions as well as for online predictions.

Again, due to the size of the original dataset, and given the time and resource limitations to develop this use case, we have decided to limit the number of variables considered in the

dataset for model training purposes, and simultaneously, to be able to consider all the dataset rows of observations for the selected features (rather than to consider more variables, but to only a sample of the features informations).

Given that we have decided to follow this strategy, we have to face the necessity to eliminate a few more variables from the dataset, in order to get to the final dataset considered for model training, validation and test.

The data pipeline adopted in this demo is composed of a Feature engineering phase (made entirely in Big Query) and saving resulting data in final tables (train validation and test tables in BigQuery, as already mentioned). Then the data written in these tables are read as Pandas dataframes so that models can be trained/ validated and tested.

Below we present a CODE SNIPPET for reading Big Query Tables in Vertex AI as Pandas dataframes

Reading train dataset in Big Query, as a Pandas dataframe, with Big query API

```
%%bigquery taxi
SELECT

        CAST(hour_trip_start as float64) as hour_trip_start,
        --day_trip_start,
        CAST(day_of_week_trip_start as float64) as day_of_week_trip_start,
        CAST(month_trip_start as float64) as month_trip_start,
        --hour_trip_end,
        --day_trip_end,
        --day_of_week_trip_end,
        --month_trip_end,
        trip_minutes,
        trip_miles,
        --fare,
        --tips,
        --trip_total,
        pickup_latitude,
        pickup_longitude,
        --dropoff_latitude,
        --dropoff_longitude,
        distance,
        --CAST(company_num as float64) as company_num
FROM
  `gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_dataset_train`;
```

```
Query is running:   0%|          |
Downloading:   0%|          |
```

Machine Learning model design and selection

in this section we present the machine learning models considered for this demo, which are deep learning neural networks models.

No special components (e.g., convolutional layers) were needed in the deep learning models architecture. Rectified Linear Activation functions were adopted in all layers for their beneficial properties in deep architectures. The Adam optimization algorithm was selected to address common issues like gradient vanishing in deep neural networks.

The model selection criteria used model selection was model performance in the validation set, according to the performance metrics adopted (MSE, r2 coefficient and explained variance). The final chosen model was the one with highest explained variance and r2 and at the same time, did not present a very high MSE.

Machine Learning models and different architectures considered

The machine learning models considered for this Demo are Deep learning neural networks models.

We have tried, along the development of this use case, different neural networks architectures , as well as different use of hyperparameters (specially the learning_rate parameter, and the number of layers considered, as well as the number of neurons in each layer).

Below we present the historic of the main different architecture used, as well as the architecture of final model chosen.

- First model architecture deployed (with name ‘seventh_deep_learning_model’ in Model Registry) was made of:

An input layer densely connected, containing Relu (rectified linear units) activation functions, with seven neurons.

A second layer with Relu units and 2 neurons,

A third layer with Relu units and 2 neurons,

An output layer with only 1 neuron.

This model used Adam optimizer, with MSE as loss function and learning rate of 0.8.

- A second model architecture considered was made of:

An input layer densely connected, containing Relu (rectified linear units) activation functions, with seven neurons.

A second layer with Relu units and 5 neurons,

A third layer with Relu units and 3 neurons,

A fourth layer with Relu units and 2 neurons,

An output layer with only 1 neuron

This model used Adam optimizer, with MSE as loss function and learning rate of 0.08.

It also included the parameters beta_1 = 0.9, beta_2 = 0.999, epsilon = None, decay = 0.0 and amsgrad = False.

- A Third and final model trained (and deployed, with name 'seventh_deep_learning_model' in Vertex) with the exact same architecture of the second one, but with learning_rate set at 0.001.

All models were trained with the exact same seven seven previously selected features, as presented in .

Code Snippets for model design and selection

Below we present a code snippet containing the Python function that computes the regression metrics used to evaluate the different predictive models in this demo. These metrics were used for model selection (we chose the model with highest r2, explained variance and lowest MSE).

CODE SNIPPET:

```python
import sklearn.metrics as metrics
def regression_results(y_train, y_train_pred):

    # Regression metrics
    explained_variance=metrics.explained_variance_score(y_train, y_train_pred)
    mean_absolute_error=metrics.mean_absolute_error(y_train, y_train_pred)
    mse=metrics.mean_squared_error(y_train, y_train_pred)
    #mean_squared_log_error=metrics.mean_squared_log_error(y_train, y_train_pred)
    median_absolute_error=metrics.median_absolute_error(y_train, y_train_pred)
    r2=metrics.r2_score(y_train, y_train_pred)

    print('explained_variance: ', round(explained_variance,4))
    #print('mean_squared_log_error: ', round(mean_squared_log_error,4))
    print('r2: ', round(r2,4))
    print('MAE: ', round(mean_absolute_error,4))
    print('MSE: ', round(mse,4))
    print('RMSE: ', round(np.sqrt(mse),4))
```

Final obtained results

The final trained model was able to get the following results in validation set:
Figure 13: final regression model performance on validation set:

```python
regression_results(y_val, prediction_val)
```

```
explained_variance:  0.273
r2:  0.268
MAE:  0.0028
MSE:  0.0002
RMSE:  0.0134
```

As we can see, the deep learning regression final model was able to explain 27.3% of the validation set variance. Which is considered a not desirable performance.

On the test set, the final regression model obtained the following results:

Figure 14: final regression model performance on test set:

```python
regression_results(y_test, prediction_test)
```

```
explained_variance:  0.2398
r2:  0.2359
MAE:  0.0027
MSE:  0.0002
RMSE:  0.0131
```

On the test set side, the deep learning regression final model was able to explain 23.98 % of the test set variance. Which is also considered a not desirable performance.

However, given the constraints that we have to deal with, these were the best possible results that it was possible to achieve for the sake of the development of this business case.

Nevertheless, in the following section we provide suggested actions to be considered as future steps in order to substantially improve the performance of the final model presented here.

Additionally, it must be said that the main purpose of this use case was to provide a complete workflow implementation of a machine learning solution to couple with presented business needs, made completely using Google cloud solutions.

We believe that this development fulfilled this purpose.

Machine learning model training and development

In this section we first present how dataset sampling was performed. The dataset was split into train, validation and test datasets, using the field trip_start_timestamp in order to do it. There was no need for random sampling because the train dataset considered data of all years involved in the raw dataset.

Regarding model training, all the models trained in this demo used the Adam optimization algorithm). Each model used a different set of hyperparameters (the learning rate) for training, in the previous section. All training sessions were conducted using the same training dataset (chicago_taxi_trips_final_dataset_train table).

We followed many Google Cloud best practices during model training, such as:
- the use of Google BigQuery for feature engineering and selection as data processing steps.
- The use of Operational training: we exported Keras models as TensorFlow artifacts, to deployment in Google Cloud endpoint.
- Artifacts organization: we used Google's Vertex AI Model Registry solution.
- We used different performance metrics to keep track/ monitor model performance.

We used mean squared error (MSE), r2 and explained variance as the evaluation metrics during training, validation and testing, giving more emphasis on MSE. MSE is considered the best metrics for regression models.This metrics is optimal for the business goal being addressed in this demo because of its main property (gives more importance to large model errors) and implying in models that give accurate predictions (in the case of the demo, the total duration of taxi trips).

Hyperparameter tuning was done along different model architectures that were considered along this demo. The main hyperparameter considered was the learning rate used for the training process. The model performance optimization was aimed to increasing r2 and explained variance while reducing MSE.
For each of these different architectures it was used an Adam optimization algorithm for training. The model with the best performance according to these metrics (on the validation set) was chosen. This was how performance optimization was accomplished.
The model bias/ variance tradeoff were determined by analyzing the models performance metrics conjointly (specially r2 coefficient, explained variance and mean squared error).

Regarding adherence to Google`s Machine learning Best Practices, several best practices were followed in this development, such as:

- **Maximization of model`s predictive precision with hyperparameters adjustments.**
- **Preparation of models artifacts to be available in Cloud Storage: this was accomplished in Demo 1 where models`s artifacts were made available in TensofFlow format (saved_model.pb files) .**
- **Specification of the number of cores and machine specifications for each project: we have defined appropriate machines (in terms of number of cores, memory and even GPU`s to be used in each Demo based on previous experience).**

Dataset sampling for model training, validation and test

The dataset sampling for model training (as well as for model validation and testing) was performed as follows: once we have created the `gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_dataset1`table, we have decided to proceed to the train, validation and test splits.

For this purpose, we have decided to split this dataset into these three subsets using the trip_start_timestamp field, in such a manner that the train dataset has approximately 70% of the data, and the validation and test datasets have respectively 20% and 10% of the data.

This splitting criteria was based on a default/ frequent rule of thumb for splitting datasets in Statistics/ Machine Learning.

In this sense, the chicago_taxi_trips_final_dataset1 was split into these three subsets such that:

- Train dataset was formed with observations ranging from trip_start_timestamp ranging from '2013-01-01 00:00:00 UTC' AND '2018-01-21 23:45:00 UTC',

- Validation dataset was formed with observations ranging from trip_start_timestamp ranging from '2018-01-21 23:45:01 UTC' AND '2018-10-24 23:45:00 UTC',

- Test dataset was formed with observations ranging from trip_start_timestamp greater than '2018-10-24 23:45:00 UTC' .

Each dataset was saved as independent tables in Big Query, named, respectively, chicago_taxi_trips_final_dataset_train, chicago_taxi_trips_final_dataset_validation and chicago_taxi_trips_final_dataset_test.

Below one presents the SQL scripts for creating these three datasets in Big Query.

Train dataset generation CODE SNIPET:

```
--- CRITERIA TO SPLITTING DE DATASET INTO TRAIN< VALIDATION AND TEST SETS
USING trip_start_timestamp as criteria



SELECT COUNT(*)/65922249 FROM
`gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_data
set1` WHERE trip_start_timestamp  BETWEEN '2013-01-01 00:00:00 UTC' AND
'2018-01-21 23:45:00 UTC';  --      0.7030748905426 train



SELECT COUNT(*)/65922249 FROM
`gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_data
set1` WHERE trip_start_timestamp  BETWEEN '2018-01-21 23:45:01 UTC' AND
'2018-10-24 23:45:00 UTC';  --  0.2019648935217 validation



SELECT COUNT(*)/65922249 FROM
`gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_data
set1` WHERE trip_start_timestamp  > '2018-10-24 23:45:00 UTC' ;  --
0.094960215935 test



--------------------------------------------------------------------------
---------------------------------



--- CREATION OF TRAIN, VALIDATION AND TEST DATASETS
```

```
--- TRAIN DATASET (70% of data)

SELECT

        --taxi_id,

        --trip_start_timestamp,

        --trip_end_timestamp,

        --payment_type,

        --company,

        minute_trip_start,

        hour_trip_start,

        day_trip_start,

        day_of_week_trip_start,

        month_trip_start,

        --year_trip_start,

        minute_trip_end,

        hour_trip_end,

        day_trip_end,

        day_of_week_trip_end,

        month_trip_end,

       -- year_trip_end,



        trip_minutes,

        trip_miles,

        --pickup_census_tract,
```

```
        --dropoff_census_tract,

        pickup_community_area,

        dropoff_community_area,

        fare,

        tips,

        --tolls,

        extras,

        trip_total,

        pickup_latitude,

        pickup_longitude,

        --pickup_location,

        dropoff_latitude,

        dropoff_longitude,

        --dropoff_location,

        distance,

        company_num,

        payment_type_num



        FROM
`gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_data
set1` WHERE trip_start_timestamp  BETWEEN '2013-01-01 00:00:00 UTC' AND
'2018-01-21 23:45:00 UTC';
```

```
--- VALIDATION DATASET (20% of data)
SELECT
        --taxi_id,
        --trip_start_timestamp,
        --trip_end_timestamp,
        --payment_type,
        --company,
        minute_trip_start,
        hour_trip_start,
        day_trip_start,
        day_of_week_trip_start,
        month_trip_start,
        --year_trip_start,
        minute_trip_end,
        hour_trip_end,
        day_trip_end,
        day_of_week_trip_end,
        month_trip_end,
       -- year_trip_end,

        trip_minutes,
        trip_miles,
```

```
        --pickup_census_tract,

        --dropoff_census_tract,

        pickup_community_area,

        dropoff_community_area,

        fare,

        tips,

        --tolls,

        extras,

        trip_total,

        pickup_latitude,

        pickup_longitude,

        --pickup_location,

        dropoff_latitude,

        dropoff_longitude,

        --dropoff_location,

        distance,

        company_num,

        payment_type_num


        FROM
`gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_data
set1` WHERE trip_start_timestamp  BETWEEN '2018-01-21 23:45:01 UTC' AND
'2018-10-24 23:45:00 UTC';
```

```
----- Test dataset creation (10% of data)


SELECT

        --taxi_id,

        --trip_start_timestamp,

        --trip_end_timestamp,

        --payment_type,

        --company,

        minute_trip_start,

        hour_trip_start,

        day_trip_start,

        day_of_week_trip_start,

        month_trip_start,

        --year_trip_start,

        minute_trip_end,

        hour_trip_end,

        day_trip_end,

        day_of_week_trip_end,

        month_trip_end,

       -- year_trip_end,
```

```
    trip_minutes,
    trip_miles,
    --pickup_census_tract,
    --dropoff_census_tract,
    pickup_community_area,
    dropoff_community_area,
    fare,
    tips,
    --tolls,
    extras,
    trip_total,
    pickup_latitude,
    pickup_longitude,
    --pickup_location,
    dropoff_latitude,
    dropoff_longitude,
    --dropoff_location,
    distance,
    company_num,
    payment_type_num
```

```
        FROM
`gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_data
set1` WHERE trip_start_timestamp  > '2018-10-24 23:45:00 UTC' ;
```

Further elimination of variables in the dataset for model training

Due to the still very large size of these datasets, and the available infrastructure available for the development of this demo, it was still necessary to further reduce the dataset size, which forced to further reducing the number of features considered for Deep Learning Model training (following the data strategy of keeping most of the rows for each feature, even that this leads to the sacrifice of reducing even further the number of variables for model training).

In this sense we further discarded the following variables for model training, validation and test:

- Day_trip_start (this feature was discarded because we already had the day_of_the_week_trip_start variable in the dataset),
- Hour_trip_end,
- Day_trip_end,
- Day_of_the_week_trip_end,
- Fare,
- Dropoff_latitude,
- Dropoff_longitude
- Company_num

With these choices, we ended up with final datasets composed of the following final features:

- Hour_trip_start,

- Day_of_week_trip_start,
- Month_trip_start,
- Trip_minutes (the target variable for the regression, Deep Learning modelo to be trained),
- Trip_miles,
- Pickup_latitude,
- Pickup_longitude,
- Distance.

Totalling 7 features and one target variable.

In summary, the split of the whole dataset into train, validation and test datasets used the field trip_start_timestamp in order to do it. There was no need for random sampling because train dataset considered data of all years involved in the raw dataset

Proposed Machine Learning Model

Given the original dataset size, containing millions of data rows, and the tabular nature of the dataset, we have decided for a deep neural network solution to build up a regression in order to keep up with the business need of predicting the total time of any given taxi trip in Chicago city.

As the data is tabular, no special component/ layer (like convolutional layers) in the proposed neural network solution were needed.

For this proposed solution, we have adopted Rectified Linear Activation functions in all layers, due to its good properties, specially when considering neural networks deep architectures.

Regarding the optimization algorithm, we proposed to use Adam, which is recommended for deep learning models, in order to approach some well known problems in deep neural networks training, such as gradient vanishing.

Used Framework

For the purposes of the development of a machine learning solution for the business problem at stake, we chose to utilize Keras framework (using Tensorflow as backend). In particular due to the constraints imposed by Google Cloud for model registry and deployment, we chose Keras version 2.12.0 (mainly because for model deployment we were restricted to specific tensorflow versions, up to version 2.12.0) .

The main reason for choosing Keras to develop a Deep Learning solution was because Keras provides an abstraction layer over tensorflow, so that model specification and training is simpler , and given time and resources limitations for the development of this case, this seemed to be the best choice.

Deep Neural Network architecture choice

The chosen final Deep Learning model architecture was the result of the influence of many factors:

- The infrastructure resources available for the development of this business case (specially regarding Memory use and computation time),
- THe size of the training dataset (given the data strategies adopted along the development of this use case).
- Time constraints for the development of this use case, especially regarding to data exploration, data strategies, feature engineering, model training,evaluation, testing, registering and deployment,
- Testing of the deployed model with batch predictions with Google Cloud solutions.

Considering these factors we trained different neural networks architectures, with varying number of layers and nodes within each layer, evaluating memory consumption and computing time mainly.

Due to the aforementioned constraints and the available infrastructure for the model training sessions with Vertex AI Workbench, we ended up with the following final neural network architecture:

- An input layer densely connected, containing Relu (rectified linear units) activation functions. This input layer receives input vectors containing observations for the seven finally selected features for the model. This layer has seven nodes.
- A second layer, densely connected, containing five nodes and also using Relu units.
- A third layer, densely connected, containing three nodes and also using Relu units.
- A fourth layer, densely connected, containing two nodes and also using Relu units.

- A final output layer, containing only one node (as this is a regression problem with only one target variable to be predicted by the input of the feature vectors) .

It's worth mentioning here that we had to consider very few nodes in the final layers due to the above mentioned constraints faced along use case developments.

The below code snippet shows the architecture of the final Deep Neural Network considered in this use case:

CODE SNIPET:

```
model = Sequential()
model.add(Dense(7, input_shape=(7,), activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(2, activation='relu'))
model.add(Dense(1))
```

```
# Compile model
optimizer = keras.optimizers.Adam(learning_rate = 0.001, beta_1 = 0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
model.compile(loss='mean_squared_error', optimizer=optimizer)
```

Hyperparameter tuning, performance optimization and training configuration

Regarding hyperparameter tuning, the main hyperparameter considered was the learning rate used for the training process. The model performance optimization was aimed to increasing r2 and explained variance while reducing MSE.

Once again, due to time and hardware limitations we were not able to implement cross validation schemas along the trainings. We also had to keep the learning rate somewhat high for the initial trainings, but, due to the poor performances faced, we carried out a final training session with an appropriate value for this rate.

Three different deep learning models were trained, each one considering a different architecture and a given learning rate. The first trained model used a learning rate of 0.8, the second one a learning rate of 0.08 and the third (chosen final) model a learning rate of 0.01. So that hyperparameter tuning was done along these three training sessions

For this matter, we ended up with a chosen learning rate of 0.01 to accomplish training of the Deep Learning model in order to have a regression model to predict the total time length of the trips (measured in minutes).

For the training of this Deep learning solution, we adopted as loss function the Mean Squared Error (MSE) and Adam as the optimization algorithm along the training.

Also, due specifically to time constraints for the development of this machine learning solution, we used 10 epochs along the final training session.

Other parameters used in the final training session:

- Beta_1: set to 0.9,
- Beta_2: set to 0.999,
- Epsilon: set to None,
- Decay: set to 0.0
- Amsgrad: set to False

These set of parameters were set in order to seek learning optimization.

Model training

All the models trained in this demo used the Adam optimization algorithm. Each model used a different set of hyperparameters (including the learning rate, and the number of epochs) for training, in the previous section. All training sessions were conducted using the same training dataset (chicago_taxi_trips_final_dataset_train table).

Adherence to Google's Machine Learning Best Practices

As presented along this document, we followed Google's Machine Learning Best Practices in the planning and implementation of all the workflow depicted in this document. In some (few) cases it was not possible to follow some Machine Learning Best Practices documentation's suggestions either by time or costs constraints; but for the very most part the implemented workflow followed this Best Practices documentation (available in the link: https://cloud.google.com/architecture/ml-on-gcp-best-practices)

First of all it must be said that, we used, in each step of the Machine Learning models developments, the products recommended by Google's ML Best Practices (Link: https://cloud.google.com/architecture/ml-on-gcp-best-practices?hl=pt-br#use-recommended-tools-and-products):

- Configuration of ML environment step: we used instances of Vertex AI Workbench for this step.
- Machine Learning Development step: we used the following products in this step:

BigQuery, Cloud Storage and instances of Vertex AI Workbench.Specifically, we used Vertex AI Workbench instances for experimentation and development of models .

- Data processing steps: we used Google BigQuery mostly for feature engineering and selection.
- Operational training: we exported Keras models as TensorFlow artifacts, to deployment in Google Cloud endpoint.
- Artifacts organization: we used Google's Vertex AI Model Registry solution.

We stored resources (like files containing test data) and model artifacts (TensorFlow model artifacts) in Cloud Storage. These artifacts and resources are stored in Cloud Storage associated within a specific project where only allowed people can have access to. Additional Identity Access Management (IAM) prerogatives can be set for each user.

- For Machine Learning Environments: we used personalized models (that is, trained in a customized way, with proper code) using Vertex AI environment.

Besides adopting the above recommended products as ML Best Practices, we also followed the Best Practices below.

- Data Preparation for training

  Observing Google Machine Learning Best Practices, training data were extracted from origin/ data sources and converted to appropriate format for machine learning training purposes (this was accomplished in feature engineering phase). Final data was stored in appropriate Google Cloud Resources (BigQuery).

- Store Structureed data in BigQuery

  Raw, intermediate and final datasets were stored in BigQuery.

- Avoid to store data in block storaging:

  We have not stored any data in block storaging style (like files in networks, or hard disks). Instead we used BigQuery.

  We also have not read any data directly from any specific database other than BIgQuery or Cloud Storage for optimal performance.

- Maximize model`s predictive precision with hyperparameters adjustments

  This was done in Demo 1.

- Prepare models artifacts to be available in Cloud Storage:

This was accomplished in Demo 1 where models`s artifacts were made available in TensofFlow format (saved_model.pb files) .

- Specify the number of cores and machine specifications for each project

  We have defined appropriate machines (in terms of number of cores, memory and even GPU`s to be used in each Demo base on previous experience training models for each demo and also taking into consideration the dataset size).

- Plan the model data entries:

We have planed how input, new test data are to be transmitted to trained final models. So that we judged that for batch predictions, for example, input data are to be stored and made available for models from Cloud Storage, for the demo.

Finally, it must be said that all models were deployed to an endpoint and containerized, using VErtx default options, with default Google Cloud containers, in Vertex Model Registry.

As stated previously we have not followed some of the Bet Practices suggestions, either because of time constraints or other factors.

So that for example, for this demo, we have not used Docker containerized models specially because of time constraints for developing this demo. Instead we used default containerization available in Google Cloud.

Another example of a point in Best practices that was not followed is the use of Datasets in Vertex.This was because of the size of the initial dataset and also because of specific instructions, for this Demo, to use Bigquery (other options were to use Dataproc or Dataflow) for for data preprocessing. So storing the initial dataset in BigQuery was straightforward in order to make data preprocessing steps easier.

Model monitoring was made for some performance metrics (specificallly the Mean Squared Error - MSE - and Explained Variance).

Bias/ Variance tradeoff

The model bias/ variance tradeoff were determined by analyzing the models performance metrics conjointly in the validation dataset.

The best model was considered the one with highest explained variance and r2 and that, at the same time did not presented very high MSE.

Given the choice of the features for model training, we have taken all possible actions to seek a good bias/ variance tradeoff. This includes:

- Considering a train dataset that is representative of all taxi trips,
- Considering appropriate learning rates,
- - We have considered models with no signs of overffiting the data.

In general one accepts the idea that if a model presents an r2 coefficient not greater than 90% the model does not overfit the data and therefore its predictions are not too close of real/observed data (so that models variance is not too high) and on principle, the model would be able to generalize well on new, unseen datasets.

The ideal scenario would be to train a final model that would have r2 coefficient (and explained variance) of around 90% on validation and test sets and a MSE neither too high or too low.

The best model achieved in this demo did not presented signs of overfitting the data , by its low r2, explained variance and MSE (instead it presented sings of underfitting the datasets).

Machine Learning Model Evaluation/ Model Performance Assessment

In this section we present how final model performed in the validation as well as in an independent test set. We also present code snippets showing how to access model performance on the test dataset.

When considering how final model performed in the test set, we see that it had not yet achieved a typical performance for production purposes, to perfectly address the business goal. The model was able to explain 23.98 % of the test set variance,also achieving a MSE of 0.0002.

Some possible reasons for this are enumerated such as that the select features for the model were not the best ones (and we should consider another set of features, for example using a feature selector algorithm). However the main goal of this demo was to provide a complete workflow to trains and deploy a model to address this goal, and this was accomplished.

We have selected the MSE (mean squared error) metric as the main one to assess model performance on the training validation and test sets.

This is due to the fact that the MSE gives more importance to large model errors, being the variance of the model residuals. Also, it is a key/ standard metric to tackle model performance both in academia as well in companies as a whole. Because of that MSE is considered the best metrics for regression models. This metrics is optimal for the business goal being addressed in this demo because of its main property (gives more importance to large model errors) and is considered in general the best metric for developing predictive models that give accurate predictions (in the case of the demo, the total duration of taxi trips).

Other metrics, such as the Mean Absolute Error (MAE), determination coefficient (R2), and the Root Mean Squared Error (RMSE) were also used in the python notebook.

These metrics were used to make an assessment of the final model's performance on validation and test sets.

We also used the same metrics to evaluate previous trained models in the provided notebooks for this Demo 1 (CHICAGO_TAXI_TRIPS_MODEL_TRAINNING_EDITED_FINAL.ipynb and EDITABLE_MODEL_APPLICATION_DEMO1.ipynb notebooks).

The final trained model was able to get the following results in validation set:

The final regression model performance on validation set as shown below:

```
regression_results(y_val, prediction_val)
```

```
explained_variance:  0.273
r2:  0.268
MAE:  0.0028
MSE:  0.0002
RMSE:  0.0134
```

As we can see, the deep learning regression final model was able to explain 27.3% of the validation set variance and achieved a MSE of 0.0002..

On the test set, the final regression model obtained the following results:

```
regression_results(y_test, prediction_test)
```

```
explained_variance:  0.2398
r2:  0.2359
MAE:  0.0027
MSE:  0.0002
RMSE:  0.0131
```

On the test set side, the deep learning regression final model was able to explain 23.98 % of the test set variance,also achieving a MSE of 0.0002.

We see from the results above, that the final model did not performed well regarding the considered metrics, in the validation and test sets. This performance typically would not be accepted for production purposes in the sense of addressing the business goal (which is to have accurate predictions for the total duration of the taxi trips.

Possible causes of this performance and solutions will be addressed in the next section. However, the main goal of the demo (which is to present a complete workflow to develop and deploy a machine learning for the sake of addressing this business problem) was accomplished.

Below we present a code snippet containing all code for assessing model performance on the independent test set:

CODE SNIPPET:

[80]:
```
## Loading new test set
```

[81]:
```
%%bigquery test_set
SELECT

        CAST(hour_trip_start as float64) as hour_trip_start,
        --day_trip_start,
        CAST(day_of_week_trip_start as float64) as day_of_week_trip_start,
        CAST(month_trip_start as float64) as month_trip_start,
        --hour_trip_end,
        --day_trip_end,
        --day_of_week_trip_end,
        --month_trip_end,
        trip_minutes,
        trip_miles,
        --fare,
        --tips,
        --trip_total,
        pickup_latitude,
        pickup_longitude,
        --dropoff_latitude,
        --dropoff_longitude,
        distance,
        --CAST(company_num as float64) as company_num
FROM
  `gcp-ml-especialization-2024.spec_taxi_trips.chicago_taxi_trips_final_dataset_test`;
```

```
Query is running:   0%|          |
Downloading:   0%|          |
```

[82]:
```
X_test = test_set.drop('trip_minutes', axis=1)

y_test = test_set[['trip_minutes']]
```

[83]:
```
del test_set
```

[ ]:
```
prediction_test = model.predict(X_test )
print("Finish making predictions on test set")
```

```
195625/195625 [==============================] - 231s 1ms/step
Finish making predictions on test set
```

[ ]:
```
regression_results(y_test, prediction_test)
```

```
explained_variance:  0.2398
r2:  0.2359
MAE:  0.0027
MSE:  0.0002
RMSE:  0.0131
```

Final considerations and recommendations for future developments

In this section we provide some suggestions, as next steps for future developments in this use case, especially regarding improving the performance obtained so far.

Due to the already presented constraints, all training sections were conducted using the exact same feature set, which showed up to be not the best feature subset for this problem.

For this reason we recommend the use of a feature selection algorithm, to be applied over a dataset sample, in order to better guide feature choice/ selection for the model.

Several good/ suitable feature selection algorithms can be used for selecting a good/ significant set of features for neural network models.

One of them is Mutual Information.

We are very confident that such procedures would be able to provide a more significant feature set to be adopted in future training.

Secondly, we could not test a large amount of Neural networks architectures in this development. In this regard, we recommend that a more vast/ broad combination of architectures should be tested, also considering more layers and different numbers of neurons in each of them.

Regarding the optimization process along the training sessions, we also encourage other combinations of hyperparameters used on it, which are key factors for good performance with deep learning model training.

The main objective of this development was to present a complete workflow of the development of a machine learning model, to fulfill a specific business need, in Google Cloud. This development provided a detailed description of all steps involved in this sense (from data exploration to model deployment), following Google Machine Learning Best Practices, as well as recommendations for future developments.

Lessons Learned

We have adopted a data strategy which corresponds to choosing a subset of the available features in the dataset, while keeping the most observations for this subset. This data strategy, combined with the hyperparameters choices and training strategies resulted in a not so good performing predictive model.

As stated before, due to time and costs constraints, we had to stop the development of thjis demo at this point.

However other data strategies (for example keeping more feature and simultaneously considering a large sample for all considered features) migh give better predictive results.

Having more available time for training might as well give better results especially considering Deep Learning Models.