The key steps of Machine Learning model development: Example of an end-to-end TensorFlow pipeline

Business goal and machine learning solution

There are many ways to start a taxi trip. You can either wait on the side of the road for one, walk to a taxi stand, or even phone an agency to book one for you. However, with the widespread use of smartphones, there is an even more convenient method - booking a trip directly from your app. As opposed to waiting for one on the street, the user no longer has to worry about whether a taxi would actually appear or not.

Therefore, there is a business need for an app which connects users with taxi drivers, to help them organise a trip. This could furthermore be leveraged by transportation agencies, which can suggest and book taxi trips to connect with public transport.

When booking a taxi trip on an app, it seems logical to provide an estimate for the cost of the taxi trip, to give more information to the user before they have made their decision. Using a data based approach in estimating the cost could provide additional benefits, such as gaining insight into customer patterns, for example, higher prices could be charged during busier hours.

With this business goal in mind, for this demo, we predict taxi fares based on the pick-up and drop-off locations, among other variables, in a case where a hypothetical user wants to book a trip to and from arbitrary locations. For this, we have developed a machine learning solution, where we trained a regression model to predict taxi fare for any given trip.

Data Exploration

For any machine learning solution, first we have to start off with the data. As for the data used in this demo, we have used the Chicago Taxi Trips dataset, which contains the information for a given taxi trip in Chicago, such as the start and end locations, the fare charged, trip duration and distance, among other things. In this section, we describe the steps taken to explore the data to understand it, along with any insights we have picked up along the way.

.2.1 Tools used and methodology

The data exploration for this method was done in Google Cloud, on their platform built for Machine Learning, Vertex AI. Specifically, the data exploration was done in Vertex AI Workbench, where notebooks could be created to run arbitrary python code snippets. The graphs generated in this section were all generated with custom code, by using visualisation libraries installed in python (such as seaborn and matplotlib). Some statistical exploration was also performed by exporting it to a Vertex AI Managed Dataset, where the **"Generated Statistics"** feature was used.

.2.2 Data overview and subset selection

The Chicago taxi trips dataset (BigQuery) consists of taxi trips in the City of Chicago from 2013 to the present, resulting in 75 GB of 200 million records over a 10-year time period.

Considering our objective of predicting taxi fares, we hypothesised that recent records will be more predictive than older records. This is then confirmed by looking at the distribution of taxi fares over the years. Specifically, we found a noticeable price increase at around 2021 as seen in the figure below. If the whole dataset was used, the price increase (change of distribution of the target variable) is significant enough to induce train serving skew if predictions were made with modern values. **Therefore, for the purposes of this demo, we have decided to restrict the dataset to include taxi trips from 2022 onwards**.

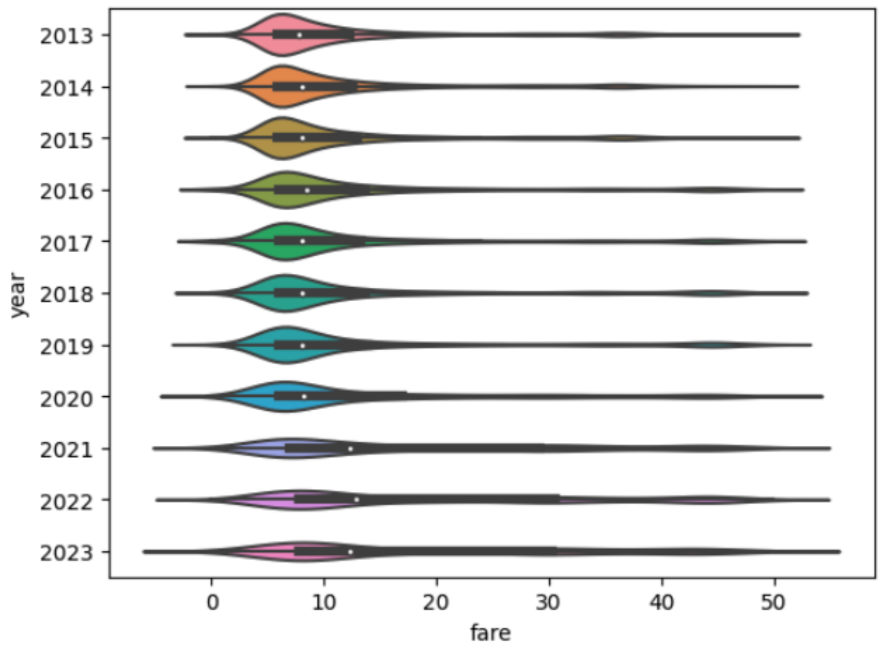
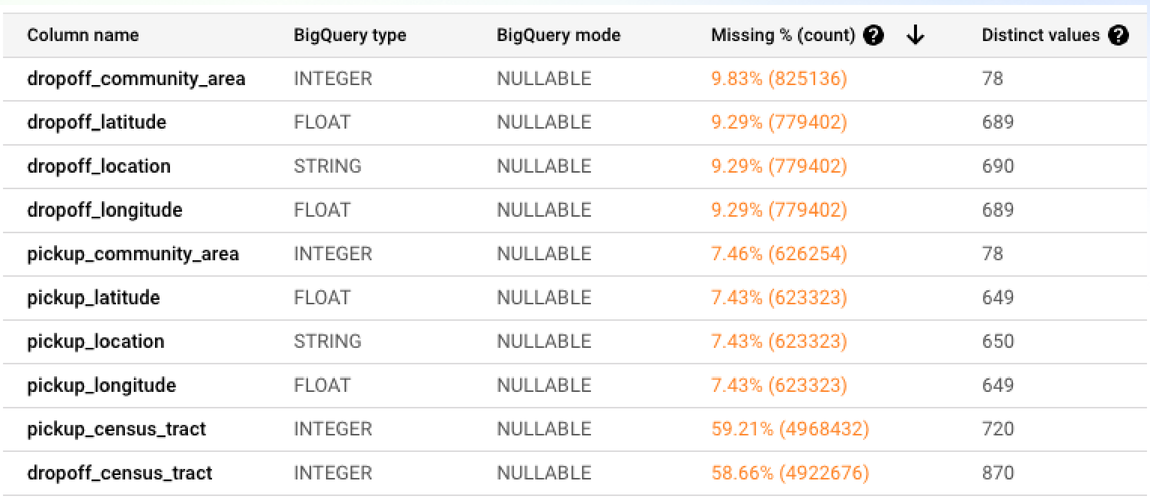


Fig.1 Taxi fare shift with Year

Simple statistical analysis

The first thing we have done was to generate a simple statistical summary of the values in the dataset. For this, we have imported the data as a dataset in Vertex AI, and used the "Generated Statistics" feature.

Upon simple inspection, we have discovered that many of the entries of the dataset contain missing values, which are listed in fig.2. Except for *pickup_census_tract* and *dropoff_census_tract*, other missing values are mostly corresponding to *pickup_location* and *dropoff_location*. Since they are critical information we need for predicting taxi fare, we exclude records missing *pickup_location* and *dropoff_location*. As for *pickup_census_tract* and *dropoff_census_tract*, we found in later analysis (fig.5) that they are highly correlated with *community_area*, therefore we exclude them from our feature pool.



Column name	BigQuery type	BigQuery mode	Missing % (count) ↓	Distinct values
dropoff_community_area	INTEGER	NULLABLE	9.83% (825136)	78
dropoff_latitude	FLOAT	NULLABLE	9.29% (779402)	689
dropoff_location	STRING	NULLABLE	9.29% (779402)	690
dropoff_longitude	FLOAT	NULLABLE	9.29% (779402)	689
pickup_community_area	INTEGER	NULLABLE	7.46% (626254)	78
pickup_latitude	FLOAT	NULLABLE	7.43% (623323)	649
pickup_location	STRING	NULLABLE	7.43% (623323)	650
pickup_longitude	FLOAT	NULLABLE	7.43% (623323)	649
pickup_census_tract	INTEGER	NULLABLE	59.21% (4968432)	720
dropoff_census_tract	INTEGER	NULLABLE	58.66% (4922676)	870

Fig.2 Missing values identified through Vertex Managed-Dataset

For the most pertinent features and the target, the distribution of the data was also plotted in Fig. 3. Many of the data points were concentrated in the smaller values. The magnitude of the numbers seem reasonable for a taxi trip in a city.

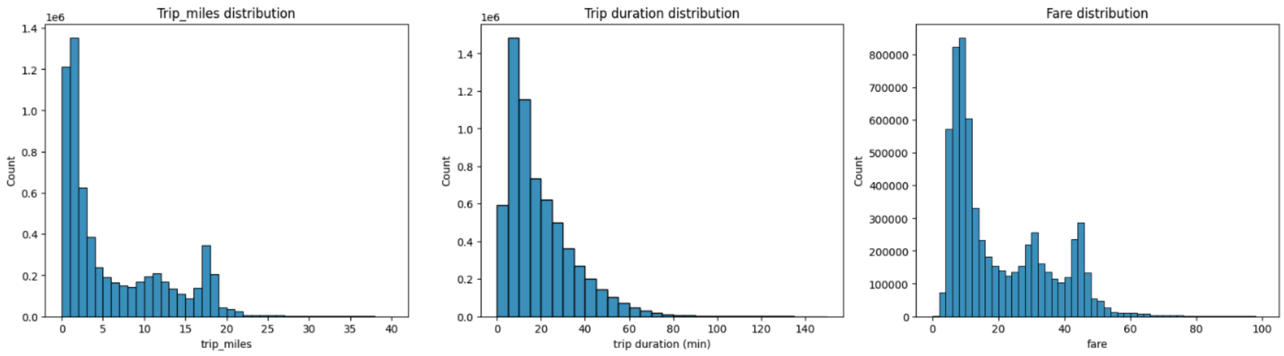


Fig.3 Trip mile, Trip duration, Fare distribution

Pickup and dropoff location distribution appears to be within reasonable range from the city centre (fig.4).

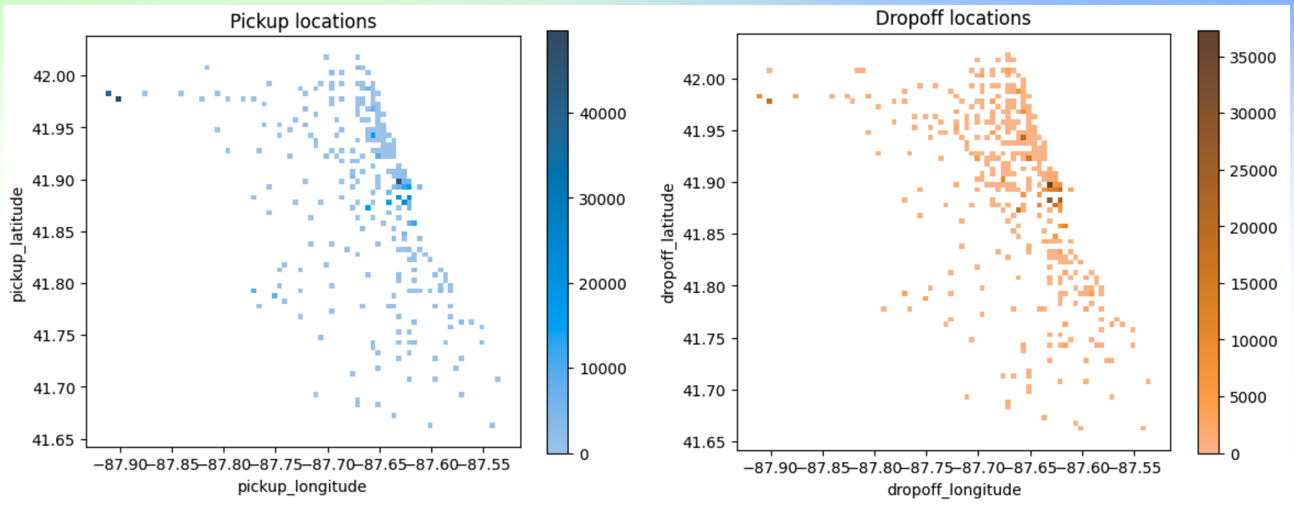


Fig.4 Pickup/Dropoff location distribution

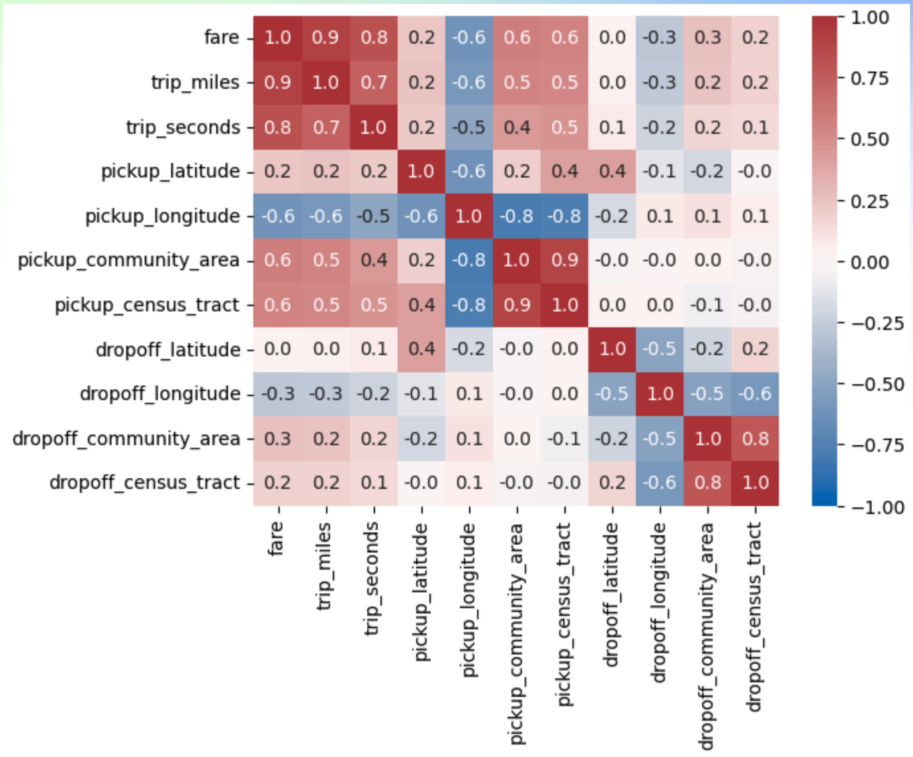
Anomalous data points

Examining the distribution of the data, we can see there are cases where we believe the data does not reflect what we would expect if it was deployed to production. Some of the important variables, such as *trip_seconds*, *trip_miles* and *fare* have data points where their values are significantly larger than it was typically expected. Therefore, we would need to discard those data points while setting an appropriate threshold such that the other typical useful data points are unaffected.

- trip_seconds has a maximum value of 86400 (exactly 1 day), which is over 100 times the mean of the trip durations. 0.4% of the dataset has trips longer than 4000 seconds
- trip_miles has a maximum of 3460 miles which is around the distance from London to New York. It is over 1000 times the mean of the dataset. 0.3% of the dataset have trip_miles larger than 30
- fare has a maximum of 9999.99 (likely to be the maximum allowable charged for a taxi trip) which is 700 times the mean. 0.6% of the dataset have fares larger than 60 so we can discard them from the dataset

Correlation across variables

According to Pearson correlation calculated between numeric feature columns we have available in the dataset, our prediction target *fare* correlates most strongly with *trip_miles*, *trip_seconds*, and noticeable correlations are also observed with *pickup_longitude*, *pickup_community_area*, *pickup_census_tract* (fig.5). In addition, *pickup_census_tract* is highly correlated with *pickup_community_area*, which makes it safe to drop *pickup_census_tract* and keep the records with missing values in this column (50%) for training with largely the same information captured by *pickup_community_area*.



	fare	trip_miles	trip_seconds	pickup_latitude	pickup_longitude	pickup_community_area	pickup_census_tract	dropoff_latitude	dropoff_longitude	dropoff_community_area	dropoff_census_tract
fare	1.0	0.9	0.8	0.2	-0.6	0.6	0.6	0.0	-0.3	0.3	0.2
trip_miles	0.9	1.0	0.7	0.2	-0.6	0.5	0.5	0.0	-0.3	0.2	0.2
trip_seconds	0.8	0.7	1.0	0.2	-0.5	0.4	0.5	0.1	-0.2	0.2	0.1
pickup_latitude	0.2	0.2	0.2	1.0	-0.6	0.2	0.4	0.4	-0.1	-0.2	-0.0
pickup_longitude	-0.6	-0.6	-0.5	-0.6	1.0	-0.8	-0.8	-0.2	0.1	0.1	0.1
pickup_community_area	0.6	0.5	0.4	0.2	-0.8	1.0	0.9	-0.0	-0.0	0.0	-0.0
pickup_census_tract	0.6	0.5	0.5	0.4	-0.8	0.9	1.0	0.0	0.0	-0.1	-0.0
dropoff_latitude	0.0	0.0	0.1	0.4	-0.2	-0.0	0.0	1.0	-0.5	-0.2	0.2
dropoff_longitude	-0.3	-0.3	-0.2	-0.1	0.1	-0.0	0.0	-0.5	1.0	-0.5	-0.6
dropoff_community_area	0.3	0.2	0.2	-0.2	0.1	0.0	-0.1	-0.2	-0.5	1.0	0.8
dropoff_census_tract	0.2	0.2	0.1	-0.0	0.1	-0.0	-0.0	0.2	-0.6	0.8	1.0

Fig. 5 Correlation across numeric variables

We found the Trip Distance and Trip Duration correlate strongly with Taxi Fare. However, as mentioned in the business goal in Section .1, we want to predict the taxi fare before a trip is made, therefore these two variables are not available at prediction time. Nonetheless, the work examining these features is not entirely meaningless if there is a way to approximately reconstruct these features.

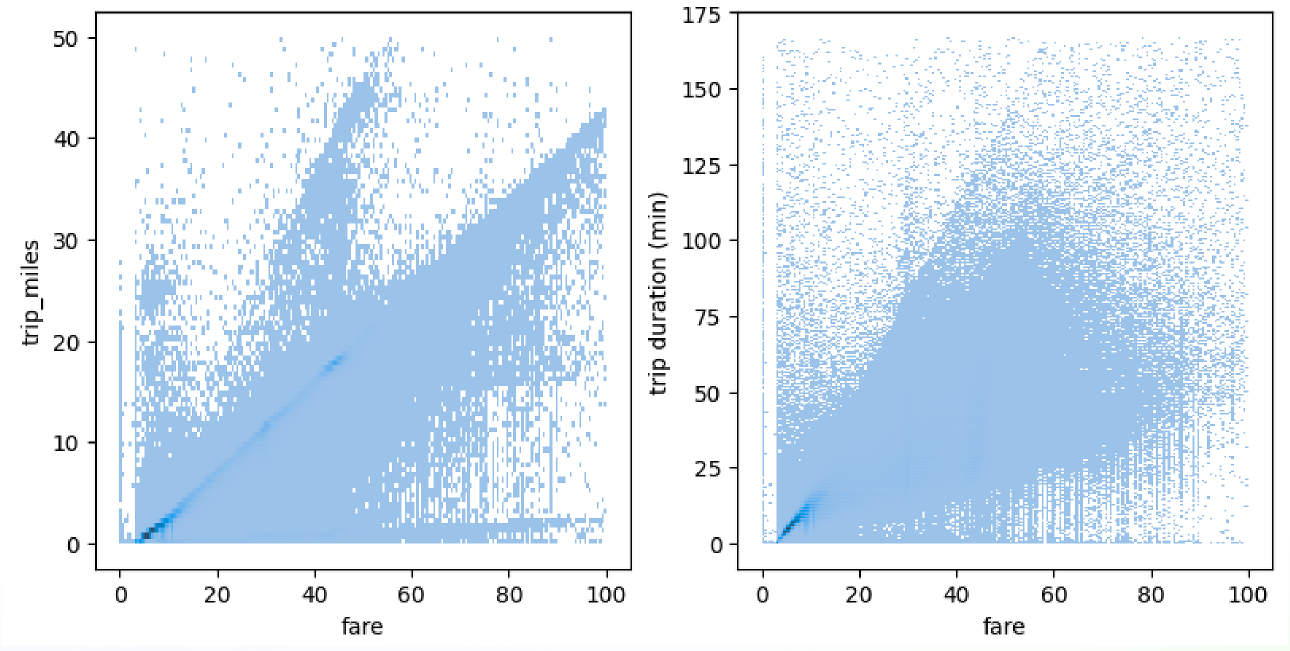


Fig.6 Linear correlation between Taxi Fare and trip miles (left) / trip duration (right)

Taxi Trip Geographic examination

Some sample trips were taken from the dataset and plotted on Google Map to analyse what a typical trip would look like, to see if we could gather any additional insight.

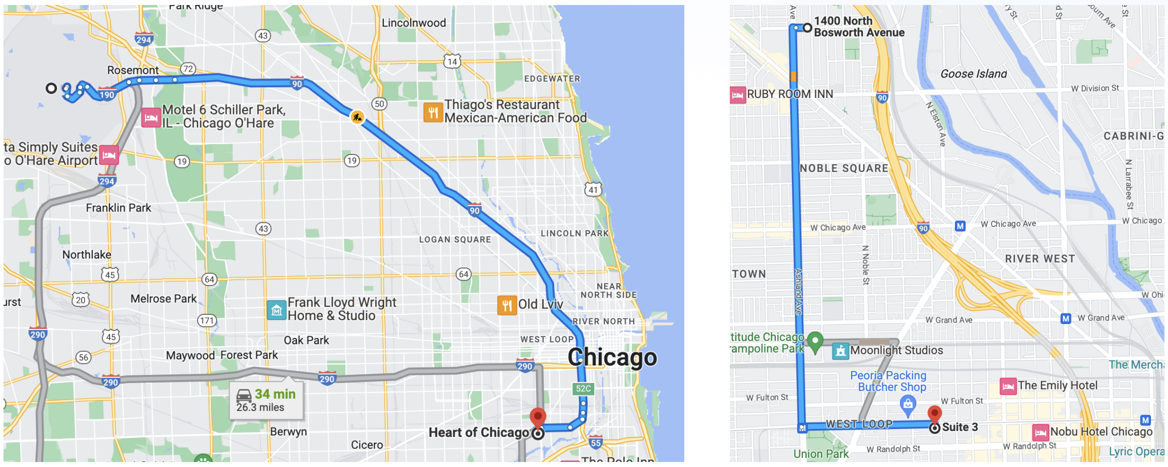


Fig.7 Example trips plotted on Google Map

As seen in fig.7, by looking at a few trips, the trips which are longer (>10 miles) seem to have a more direct route using the roads, and the shorter trips (<5 miles) seem to follow a grid based path. This is useful, as one possible way to estimate the trip distance is, for shorter trips use Manhattan distance and for longer trips use the euclidean distance. However, this pattern might not hold for some cases, as this pattern obviously depends on the location of start and end points as in some cases there are direct diagonal roads.

Decision summary:

We chose to train a regression model as we discovered that some variables in the dataset (such as trip_miles) are strong predictors of the taxi fare charged. Although trip_miles was not used in the training (as it was not available in test time), we were confident that we were able to estimate this within reasonable accuracy. To avoid large errors, we select the model architecture, based on the model's RMSE performance.

We also discovered significant outliers in the dataset which could introduce problems in the training dataset. Therefore, we decided to remove entries of the dataset where any of its values are outside acceptable ranges.

Feature engineering

The most pertinent features are picked to be used in the machine learning model are as such:

- pickup_latitude,
- pickup_longitude,
- dropoff_latitude,
- dropoff_longitude,
- company
- year,
- month,
- dayofweek,
- hour,
- euclidean distance,
- manhattan distance

Most of these variables are brought forward as is, except for the distance estimations which need further computation. As described in .2.5, both the euclidean distance and Manhattan distance can be used to estimate the trip distance, where Manhattan distance is more useful in shorter trips and Euclidean distance is more useful in longer trips, with many exceptions where direct roads are available. With this in mind, both distance measurements are brought forwards for training. In theory, the machine learning model can learn the regions where direct roads are available, and the crossover distance at which the trips stop becoming like a grid.

Preprocessing and the data pipeline

As the entire dataset is available on BigQuery, it is the most logical to do all the data preprocessing in BigQuery as well. All the data transformation and computation logic is simple enough to fit in a single query:

Unset
```
create or replace table {project}.{dataset}.taxi_clean as
select
  trip_seconds,
  trip_miles,
  pickup_latitude,
  pickup_longitude,
  dropoff_latitude,
  dropoff_longitude,
  trip_start_timestamp,
  extract(hour from trip_start_timestamp)           as hour,
  extract(dayofweek from trip_start_timestamp)      as dayofweek,
  extract(month from trip_start_timestamp)          as month,
  extract(year from trip_start_timestamp)           as year,

  -- Euclidian distance
  ST_DISTANCE(
    ST_GEOGPOINT(pickup_longitude,pickup_latitude),
    ST_GEOGPOINT(dropoff_longitude,dropoff_latitude)
  )/1000.
                                                    as euclidian,
  --convert diff to radians and multiply by radius of earth to get Manhattan
distance
  (
  abs(pickup_longitude-dropoff_longitude)+
  abs(pickup_latitude-dropoff_latitude)
  ) * 3.1415 / 180 * 6371                           as manhattan,

  fare

from `{source_table}`
where 1=1
    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 Fare > 0
    and Fare < 60
    and trip_miles > 0
    and trip_miles < 30
    and trip_start_timestamp >= '2022-01-01'
    and trip_seconds < 4000
    --non zero estimate for trip distance
    and abs(pickup_longitude-dropoff_longitude)
      +abs(pickup_latitude-dropoff_latitude) > 0
```

During training, this data preprocessing procedure is implemented in the machine learning pipeline via Vertex AI pipelines. This means that the data processing will automatically run every time before the machine learning procedure starts without any manual intervention.

Machine learning model design(s) and selection

As the goal is to predict the fare of a given trip, this machine learning model is working on a tabular dataset. As this is not an image, video, nor a forecasting dataset, there are no local correlations to take advantage of. This means architectures specific to these tasks, such as Convnets and RNNs are instantly ruled out.

We have narrowed down possible designs of the model to three options - a simple linear regression model, a deep neural network (DNN) , and a boosted tree regression model.

As BigQuery ML (BQML) could train and evaluate machine learning models with minimal effort, it was used to rapidly prototype and test out many different model architectures. A machine learning model was trained for each model architecture on BQML, with its performance on the validation dataset compared with one another:

Model Architecture	Root Mean Squared Error (RMSE)
Linear regression	4.25
Deep neural network (DNN)	4.06
Boosted tree regressor	3.86

As seen above, although it took the longest to train, the boosted tree has the best performance out of all the models. Therefore, it was selected to be the chosen model architecture.

Machine learning model training and development

Once we have decided on the model architecture used, we train the model on Vertex AI. We leverage the services of Vertex AI to develop and train the ML model.

.6.1 Data sampling

As described in the EDA section (.2.2), we decide to use the taxi trip data from 2022 onwards because of its recency and for rapid demo of concept.The selected data is then split into 3 smaller datasets, the Training set, the Validation set, and the independent Testset. The independent Testset is selected to best represent the production data distribution, therefore it has been selected to the most recent records - all records from April 1 2023 onwards. The Validation set contains records from March 1 2023 to April 1 2023, and the Training set contains all records from January 1 2022 to March 1 2023.

.6.2 Model training implementation

To ensure we follow the **Google Cloud best practices** and maximise the reusability and flexibility of our implementation, **we used Vertex AI Pipelines to orchestrate the ML workflow**.

Distribution: We trained a model within a **notebook** instance **for experimentation** first. Then we **containerized** our custom training code and its dependencies through a python distribution and submitting a **training job to Vertex AI**. In this way our training implementation is also ready for high-performance or distributed training if needed.

Device usage for storage and training: We stored our data and model artefacts in **Bigquery** and **Cloud Storage** for best integration with Vertex AI. As the model size and dataset is not prohibitively large, we selected a simple **n1-standard-4** device for training.

Monitoring: Model monitoring is enabled when deploying the custom model to an endpoint via console.

Specifically, the pipeline first runs preprocessing on Bigquery, and then uses Bigquery to split into train and test sets. Afterwards, the training job is run, with built in hyperparameter tuning. Once the best model is selected (after hyperparameter tuning), it is uploaded to the model registry, and finally evaluation is done on the test set. This entire process is simply by triggering the pipeline through a single API call.

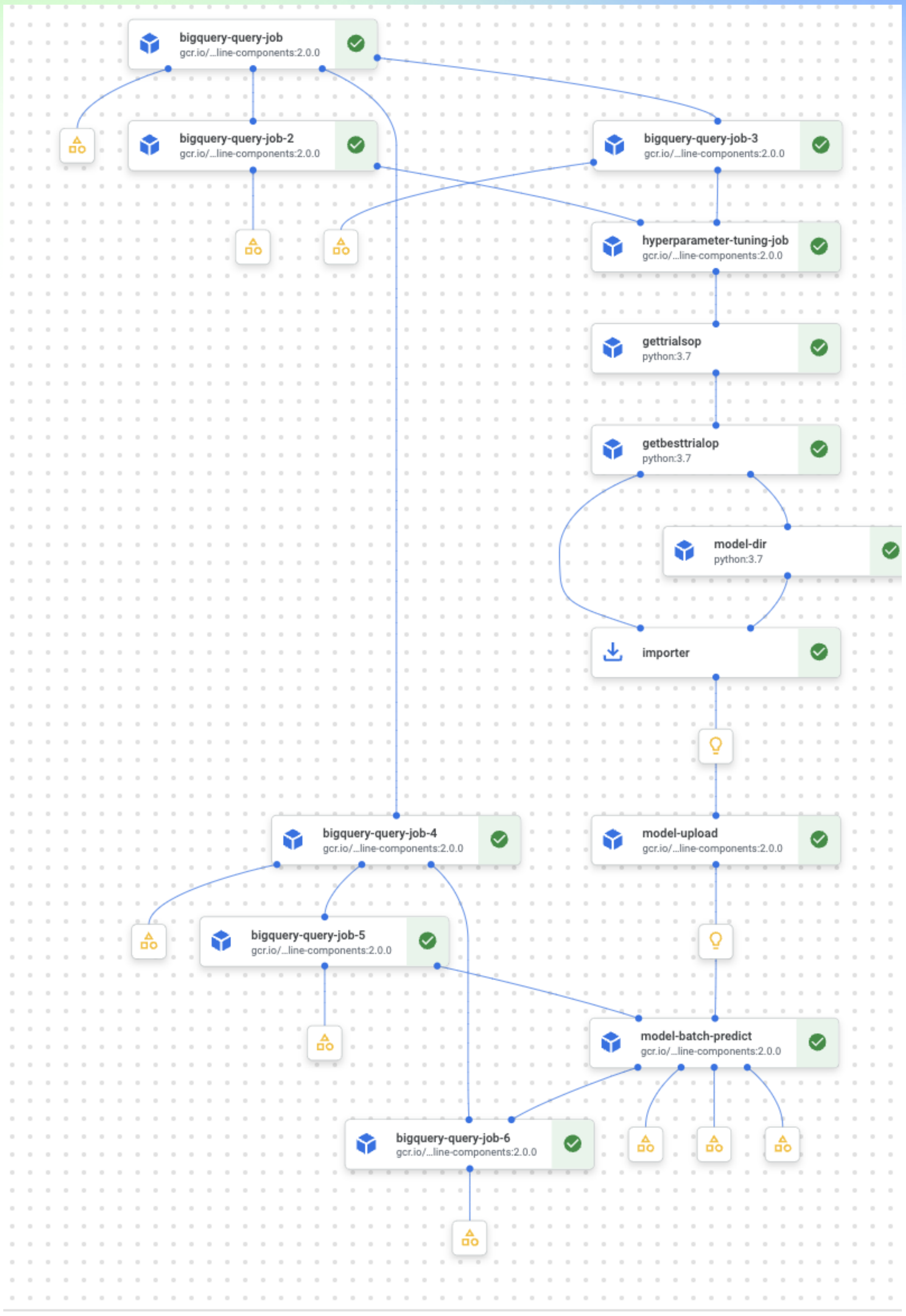

bigquery-query-job
gcr.io/...line-components:2.0.0
bigquery-query-job-2
gcr.io/...line-components:2.0.0
bigquery-query-job-3
gcr.io/...line-components:2.0.0
hyperparameter-tuning-job
gcr.io/...line-components:2.0.0
gettrialsop
python:3.7
getbesttrialop
python:3.7
model-dir
python:3.7
importer
bigquery-query-job-4
gcr.io/...line-components:2.0.0
model-upload
gcr.io/...line-components:2.0.0
bigquery-query-job-5
gcr.io/...line-components:2.0.0
model-batch-predict
gcr.io/...line-components:2.0.0
bigquery-query-job-6
gcr.io/...line-components:2.0.0

Fig.8 End-to-end Vertex AI Pipeline

.6.3 Evaluation metric

As described previously, the business goal is to provide an estimate for the taxi fare for a given taxi trip. We would want this estimate to be as accurate as possible, in other words, as close to the ground truth as possible. As a single bad estimate is more damaging to the customer experience than multiple small errors, we have decided to use RMSE as our main evaluation metric, as it disproportionately penalises large errors. Nonetheless, we also compute other evaluation metrics such as MAE, MAPE, and RMSLE in case we can gain other insight, although they are not involved in any way in the training process.

.6.4 Model performance optimisation

Hyperparameter tuning is done on the model to optimise for its performance, to minimise the squared error between its predictions and ground truth. This could be automatically done on Google Cloud, by using their Hyperparameter Tuning services.

```python
hptune_job = HyperparameterTuningJobRunOp(
  display_name = 'HPT_job',
  project = PROJECT_ID,
  location = REGION,
  base_output_directory = PIPELINE_ROOT,
  worker_pool_specs = [{
    'machine_spec':{"machine_type": 'n1-standard-4'},
    'replica_count':1,
    'python_package_spec':{
      "executor_image_uri": TRAIN_IMG,
      "package_uris": ['gs://ml-spec-us/dist/trainer-0.1.tar.gz'],
      "python_module": 'trainer.task'
    }
  }],
  study_spec_metrics =  serialize_metrics({'val_loss': 'minimise'}),
  study_spec_parameters = serialize_parameters({
    'hidden-layers': hyperparameter_tuning.IntegerParameterSpec(min=4, max=128,
scale='linear'),
  }),
  max_trial_count = 2,
  parallel_trial_count = 2,
  study_spec_algorithm = 'ALGORITHM_UNSPECIFIED'
).after(bq_train).after(bq_valid)
```

This could be seamlessly integrated into our training pipeline, as there is already a pipeline component available called HyperparameterTuningJobRunOp. This pipeline component is automatically triggered by the pipeline, and will use Vertex AI's hyperparameter tuning services to optimise the performance of the model.

Bias and Variance

Bias-variance curve (fig.9) tracks RMSE on the training set (blue) versus the validation set (red) during training iterations. For the given dataset, close-to-zero variance could possibly be due to highly adequate data size with respect to relatively simple patterns to be captured by the model. However, there is a non negligible bias present in both train and validation. This could be a result of the features being used not being a fully accurate predictor of omitted important variables such as trip_seconds and trip_miles, inducing lots of noise into the dataset. Despite this, the current RMSE indicates a prediction error below $4, which we consider within a reasonable amount given the business use case.

In this case, our current shallow boosted trees architecture is enough and shows no overfitting towards the training set. A slightly deeper architecture (with larger maximum tree depth) might help bring down the bias even further, considering we rely on the model to figure out trip_duration and trip_miles instead of directly providing the information. However, increasing the tree depth too much might make the training time too long, and compromise performance on the independent test set.

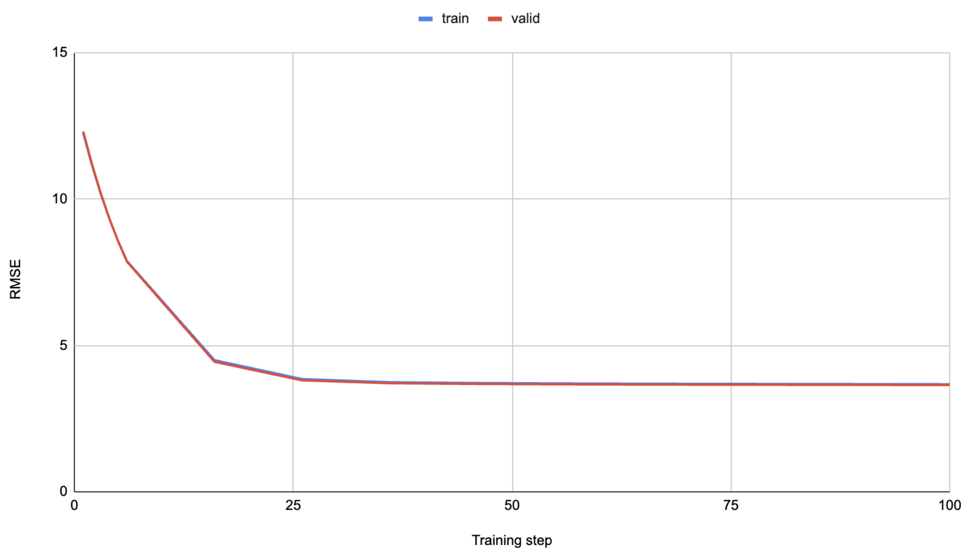


Fig.9 Bias-variance curve for training and validation

.7. Machine learning model evaluation

Independent testset reflecting production environment

As defined in the Data Sampling section in .6, distribution of the independent test set and training set are shown below (top row: test set, bottom row: training set). Distribution of key features (*trip_miles*, *trip duration*, *pickup long/lat*, and *dropoff long/lat*) and prediction target (*fare*) are consistent between test set and training set, and both reflect the production environment as they are the latest taxi trip records.

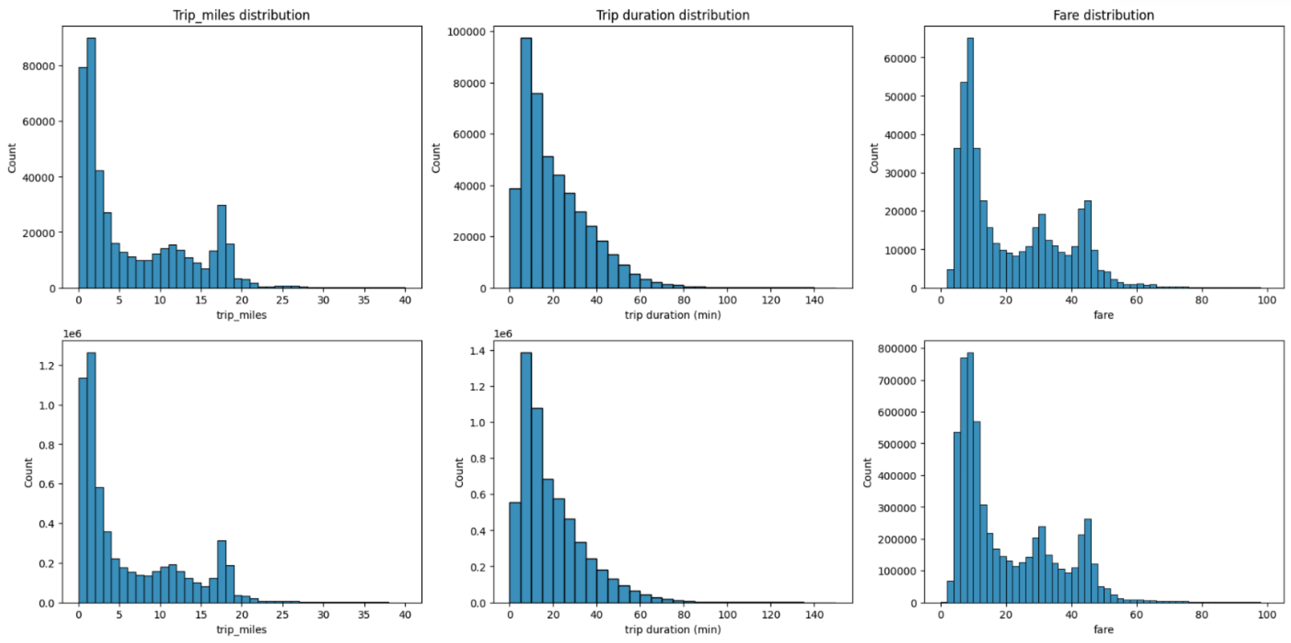


Fig.10 Trip miles, Trip duration, and Fare distributions: Production vs. independent Testset (top row: Testset, bottom row: Production)

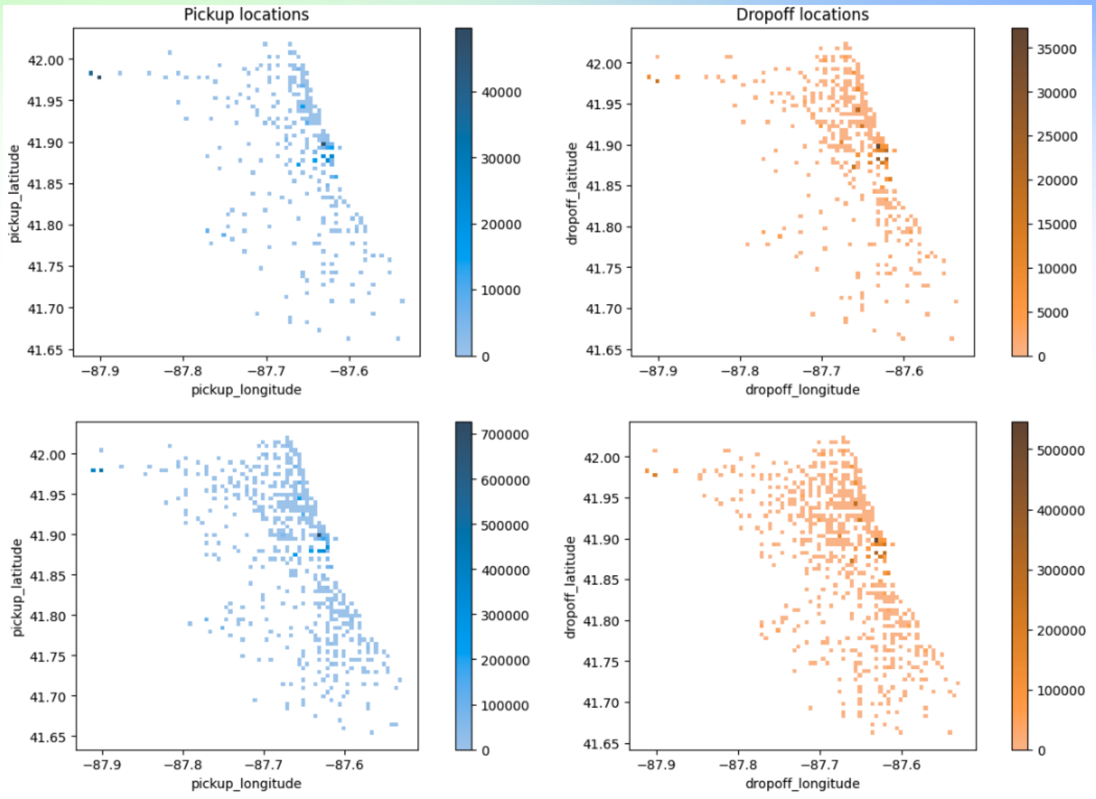


Fig. 11 Pickup and Dropoff locations distribution: Production vs. independent Testset (top row: Testset, bottom row: Production)

Model performance on independent test set

Evaluations were made on the independent test set by first running predictions through it. This is done via the Batch Prediction functionality of Vertex AI, which writes the predictions into a BigQuery table.

Afterwards, through the use of Bigquery, the predictions and the ground truth are collated, compared, and then aggregated to produce metrics

```
Unset
CREATE OR REPLACE TABLE {dataset}.taxi_evalmetrics AS(
with preds as (
  select distinct
    pickup_latitude,
    pickup_longitude,
    dropoff_latitude,
    dropoff_longitude,
    hour,
    dayofweek,
    month,
    year,
    manhattan,
    euclidian,
    round(cast(TRIM(prediction, '[]') as numeric),3) as pred_fare
  from `{dataset}.taxi_outpred`
),
preds_v_gt as (
  select
    gt.fare,
    p.pred_fare
  from `{dataset}.taxi_test` gt
  join preds p
    on gt.pickup_latitude = p.pickup_latitude
    and gt.pickup_longitude = p.pickup_longitude
    and gt.dropoff_latitude = p.dropoff_latitude
    and gt.dropoff_longitude = p.dropoff_longitude
    and gt.hour = p.hour
    and gt.dayofweek = p.dayofweek
    and gt.month= p.month
    and gt.year= p.year
    and gt.manhattan = p.manhattan
    and gt.euclidian = p.euclidian
)
select 'RMSE' as metric, sqrt(avg(pow(fare-pred_fare,2))) as value
from preds_v_gt union all
select 'MAE',avg(abs(fare-pred_fare))
from preds_v_gt union all
select 'RMSLE',sqrt(avg(pow(log(fare+1)-log(pred_fare+1),2)))
from preds_v_gt union all
select 'MAPE',avg(abs(1-pred_fare/fare))
from preds_v_gt
```

The model was able to predict taxi fares fairly accurately with an RMSE of 3.81. Upon simple inspection, most of the predictions are in line with what is expected. Other metrics, such as MAE, are listed below:

Model Architecture	Root Mean Squared Error (RMSE)	Mean Absolute Error (MAE)	Mean Absolute Percentage Error (MAPE)
Boosted tree regressor	3.813	2.195	0.156

The distributions of the RMSE are what is desired (fig.12), large errors are occurring less frequently than small errors. As seen in the chart, most of the prediction errors are below $4. However, the maximum error is 52.72. This indicates that there are either still significant outliers in the testing dataset, or that there are variables that have been significantly overlooked.

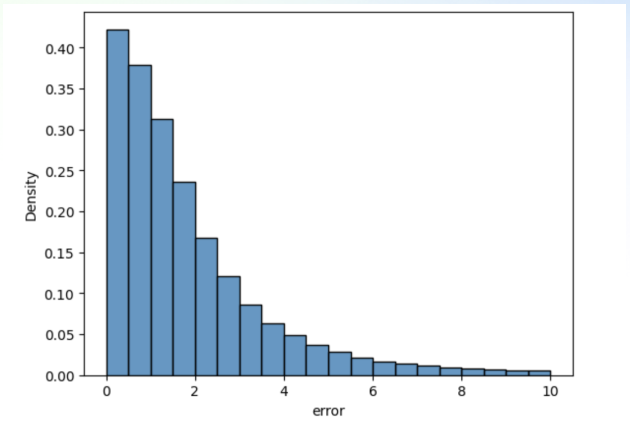


Fig.12 RMSE distribution