Whitepaper – Demo 1

Link to code repository: [btgrbo/GCP\_ML\_Demos (github.com)](https://github.com/btgrbo/GCP_ML_Demos)

Project name: bt-int-ml-specialization

Project ID: 738673379845

1. Introduction

In the dynamic landscape of urban transportation, leveraging technology for competitive advantage is key. This whitepaper presents an innovative approach implementing an end-to-end TensorFlow pipeline on the Google Cloud Platform (GCP) to optimize taxi fare predictions in Chicago based on the [Chicago taxi trips dataset](https://www.kaggle.com/chicago/chicago-taxi-trips-bq). While the core of our project involves developing a Deep Neural Network (DNN) for fare prediction, the crux of this paper is the sophisticated MLOps framework that enables such machine learning endeavours.

Our system harnesses critical data points—pickup and drop-off coordinates, timestamps, and distance—to feed into a DNN model. However, the spotlight is on the orchestration of this process within GCP's ecosystem. We demonstrate how Vertex AI, Dataflow and other GCP services are leveraged not just for model training and evaluation, but more importantly, for creating a scalable, efficient, and robust MLOps pipeline.

The integration of our model into the operational workflow of a taxi company is envisioned to be a hallmark of adaptive pricing strategies. The model’s predictions inform a dynamic pricing engine, which is part of a broader, automated MLOps pipeline. This pipeline ensures real-time responsiveness to market conditions, enhancing the agility of pricing decisions. The seamless automation in model retraining, deployment, and monitoring exemplifies the power of a well-architected MLOps infrastructure.

In this whitepaper, we delve into the intricacies of building and managing this MLOps infrastructure in GCP. We aim to provide a blueprint for effectively using cloud-based tools and services to deploy machine learning models in a production environment. The ensuing sections will detail the MLOps architecture, the challenges we navigated, and the best practices we established, offering valuable insights for organizations aspiring to harness the potential of MLOps in operational optimization.

1. Business Goal: Optimizing Pricing Strategy Through Predictive Analytics

The primary business goal of our project is to revolutionize the way taxi fares are determined and optimized, harnessing the capabilities of machine learning within the robust framework of GCP's MLOps infrastructure. In an industry where pricing decisions can significantly impact both market competitiveness and profitability, our goal is to empower the taxi company with a data-driven, dynamic pricing model.

Enhancing Revenue Management: Central to our business objective is the enhancement of revenue management. The traditional approach to taxi fare calculation, often a fixed rate based on distance and time, does not account for fluctuating demand, traffic patterns, or other situational variables. Our solution aims to integrate these dynamic factors, enabling the company to adjust fares in real-time. This responsiveness to market conditions can lead to an increase in revenue, particularly during peak demand times or in high-demand zones.

Operational Efficiency and Scalability: Implementing an MLOps-based approach also addresses the goal of operational efficiency. Automating the model training, deployment, and monitoring processes reduces manual intervention, minimizes errors, and speeds up response times. Scalability is a key consideration; as the company grows and data volumes increase, the infrastructure is designed to scale seamlessly, ensuring that the fare prediction system remains robust and reliable.

Long-Term Strategic Vision: Finally, this project aligns with the company's long-term strategic vision of being a data-centric organization. By adopting advanced analytics and machine learning, the company is not just optimizing for the present but also building a foundation for future innovations in areas like personalized customer experiences, operational optimization, and expanded service offerings.

In conclusion, the business goal of this project extends beyond the technical accomplishment of developing a predictive model. It is about transforming the core pricing strategy of the taxi company, making it more adaptive, efficient, and forward-looking. This shift towards a more data-driven approach is expected to yield significant competitive advantages and set a new standard in the urban transportation industry.

1. Machine Learning Solution: Deep Neural Network for Enhanced Fare Prediction

At the heart of our project lies a Deep Neural Network (DNN), designed to predict taxi fares with unprecedented accuracy and efficiency. This model represents a significant leap in our capability to process and interpret complex, multidimensional data. The model’s hyperparameter, learning rate and dropout, were determined by a hyperparameter tuning outlined in section Hyperparameter Tuning.

The model handles both continuous as well as categorical variables. Categorical variables include 'day\_of\_week', 'start\_month', 'start\_day', and 'start\_hour' while 'trip\_seconds', 'trip\_miles', 'pickup\_latitude', 'pickup\_longitude', 'dropoff\_latitude', and 'dropoff\_longitude' represent continuous variables. Both types of variables undergo preprocessing which is detailed in the section Preprocessing.

Our DNN consists of multiple layers, an input layer, three hidden layers with 64, 32, and 16 units respectively, and an output layer with a single unit for regression output. To prevent overfitting, each hidden layer is subject to dropout. We employ the non-linear activation function ReLU (Rectified Linear Unit) in the hidden layers to introduce non-linearity into the model, enabling it to capture complex patterns in the data. The model uses the ADAM optimizer, known for its efficiency in handling sparse gradients and its adaptiveness in different problem settings. The loss function is tailored for regression, focusing on minimizing the mean squared error in fare prediction.

The DNN is trained on a substantial subset of the Chicago Taxi Driver dataset, ensuring a comprehensive learning process. We use a split of training and validation data (80%/20%) to fine-tune the model parameters and prevent overfitting. The model's performance is evaluated using the mean-squared-error metric, ensuring its predictions are as close to the actual fares as possible. For training we employ early stopping. Specifically, we set the number of epochs to 100 but asked the algorithm to stop once the validation loss of a given epoch has been higher than the last minimum value for 10 consecutive epochs. After this criterion is reached, the model weights from the epoch with the minimum monitored validation loss are restored.

Once trained, the DNN model is integrated into the taxi company's pricing system, providing real-time fare predictions. This integration represents a significant enhancement over traditional fare calculation methods, offering a more responsive, data-driven approach to pricing.

Moving forward, we plan to continually refine the DNN model by incorporating more granular data, exploring advanced neural network architectures, and implementing continuous learning mechanisms to adapt to changing market dynamics.

In conclusion, the deployment of this DNN model marks a pivotal advancement in utilizing deep learning for practical, business-critical applications. It stands as a testament to the potential of AI in transforming industry practices, offering a glimpse into the future of intelligent transportation systems.

1. Data Exploration and Data Quality

In the first step, data was loaded into a table in BigQuery (bt-int-ml-specialization.demo1.taxi\_trips). In BigQuery Studio, we then tested for any duplicate entries by comparing the total row count to the distinct unique\_key counts and found no difference (Code Snippet 1). Hence, the dataset does not include any duplicate entries.

(1)

SELECT

COUNT(DISTINCT unique\_key) as distinct\_entries,

count(\*) as n\_rows

FROM `bt-int-ml-specialization.demo1.taxi\_trips`;

Next, we explored the variables included in the dataset: unique\_key, taxi\_id, trip\_start\_timestamp, trip\_end\_timestamp, trip\_seconds, trip\_miles, pickup\_census\_tract, dropoff\_census\_tract, pickup\_community\_area, dropoff\_community\_area, fare, tips, tolls, extras, trip\_total, payment\_type, company, pickup\_latitude, pickup\_longitude, pickup\_location, dropoff\_latitude, dropoff\_longitude, dropoff\_location.

Since our business goal is to predict taxi fare prices in real-time, we chose the variable ‘fare’ as our target. While there are many more variables which collect information about the costs associated with a ride, i.e., tips, tolls, extras, and trip\_total, the variable ‘fare’ represents the main source of income and thus also the adjusting screw for our dynamic pricing.

We then eliminated variables from the dataset which do not serve our business goal. These variables include:

* unique\_key: The unique identifier for the trip serves as primary key for the table but does not carry predictive information for the fare.
* taxi\_id: The taxi\_id is a unique identifier for the taxi. The taxi\_id might contain predictive information for the fare since a taxi\_id is tied to a specific company with a specific pricing strategy. A given taxi\_id will also be associated with a number of seats and a certain level of fuel consumption. Yet, this information is extremely sparse since the dataset contained approximately 10.000 different taxi\_ids (Code Snippet 2). We thus did not use the taxi\_id as input to our model even though future, fine-tuned approaches might as well consider it.

(2)

SELECT

  COUNT(DISTINCT taxi\_id) as distinct\_taxis,

  COUNT(DISTINCT unique\_key) as distinct\_trips

FROM `bt-int-ml-specialization.demo1.taxi\_trips`;

* trip\_end\_time: Because we considered trip\_start\_timestamp and trip\_seconds, we argue that trip\_end\_time should not deliver additional value to our prediction.
* pickup/dropoff\_census\_tract: While pickup and drop-off location will certainly explain variability in the target variable, the census tract variables yielded a high percentage of missing values (~37%)(Code Snippet 3). This can be explained by the fact that for some trips census tract is not included in the data for privacy reasons. As a measure for pickup and drop-off location we hence focused on the latitude and longitude coordinates of the center of the census tract.

(3)

SELECT

  (count(\*) - count(trip\_start\_timestamp)) / count(\*) \* 100 as perc\_missing\_trip\_start\_timestamp,

  (count(\*) - count(trip\_end\_timestamp)) / count(\*) \* 100 as perc\_missing\_trip\_end\_timestamp,

  (count(\*) - count(pickup\_census\_tract)) / count(\*) \* 100 as perc\_missing\_pickup\_census\_tract,

  (count(\*) - count(dropoff\_census\_tract)) / count(\*) \* 100 as perc\_missing\_dropoff\_census\_tract,

  (count(\*) - count(pickup\_community\_area)) / count(\*) \* 100 as perc\_missing\_pickup\_community\_area,

  (count(\*) - count(dropoff\_community\_area)) / count(\*) \* 100 as perc\_missing\_dropoff\_community\_area,

  (count(\*) - count(fare)) / count(\*) \* 100 as perc\_missing\_fare,

  (count(\*) - count(trip\_seconds)) / count(\*) \* 100 as perc\_missing\_trip\_seconds,

  (count(\*) - count(trip\_miles)) / count(\*) \* 100 as perc\_missing\_trip\_miles,

  (count(\*) - count(pickup\_latitude)) / count(\*) \* 100 as perc\_missing\_pickup\_latitude,

  (count(\*) - count(pickup\_longitude)) / count(\*) \* 100 as perc\_missing\_pickup\_longitude,

  (count(\*) - count(dropoff\_latitude)) / count(\*) \* 100 as perc\_missing\_dropoff\_latitude,

  (count(\*) - count(dropoff\_longitude)) / count(\*) \* 100 as perc\_missing\_dropoff\_longitude

FROM `bt-int-ml-specialization.demo1.taxi\_trips`;

* pickup/dropoff\_community\_area: The community area represents another variable describing the start and end location of the trip. However, compared to the census tract it is less fine-grained since it comprises a larger area. We hence focused on the latitude and longitude coordinates of the center of the census tract.
* tips: We did not consider tips. The size of the tip is information that is only available after the trip has ended and is therefore not suited for our model that should enable us to find a right price before the trip starts.
* tolls: We did not consider tolls because in 99.9% of the trip’s tolls were either 0 or missing (Code Snippet 4).

(4)

SELECT

  COUNTIF(tolls IS NULL) AS missing\_count,

  COUNTIF(tolls = 0) AS zero\_count,

  COUNT(\*) AS total\_count,

  (COUNTIF(tolls IS NULL) + COUNTIF(tolls = 0)) / count(\*) \* 100 as perc\_zero\_missing

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`;

* extras: Similarly, extras was 0 or missing for 63% of the trips (Code Snipet 5). This together with the fact that little information was provided with respect to what these extras included, we excluded it from the inputs to the model.

(5)

SELECT

  COUNTIF(extras IS NULL) AS missing\_count,

  COUNTIF(extras = 0) AS zero\_count,

  COUNT(\*) AS total\_count,

  (COUNTIF(extras IS NULL) + COUNTIF(extras = 0)) / count(\*) \* 100 as perc\_zero\_missing

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`;

* trip\_total: Because our target variable was fare and none of the other variables included in trip\_total (tips, tolls, and extras) was of significance, we excluded it.
* payment\_type: Like tips, payment type is only available at the end of the trip and therefore does not enable us to set the fare for the trip.
* company: We did not consider information regarding the taxi company since our use case aims to provide a dynamic pricing model for one specific company.
* pickup/dropoff\_location: Because we included pickup/dropoff/longitude/latitude in our model, we discarded pickup/dropoff\_location as these variables represent redundant information.

Based on these exclusions we were left with a table containing 8 fields: trip\_start\_timestamp, trip\_seconds, trip\_miles, fare, pickup\_latitude, pickup\_longitude, dropoff\_latitude, dropoff\_longitude.

For these fields we investigated the percentage of missing values (Table 1). We found that for the fields trip\_start\_time, fare, trip\_seconds and trip\_miles missing values occurred rarely (<1%), while pickup/dropoff/latitude/longitude were missing in 11,41% and 13,26% respectively (Code Snippet 3).

|  |  |
| --- | --- |
| Field | Missing Values (%) |
| trip\_start\_time | 0,00 |
| fare | 0,01 |
| trip\_seconds | 0,62 |
| trip\_miles | 0,00 |
| pickup\_latitude | 11,41 |
| pickup\_longitude | 11,41 |
| dropoff\_latitude | 13,26 |
| dropoff\_longitude | 13,26 |

Table 1 Percent of values missing per field.

For the float variables fare, trip\_seconds, and trip\_miles we expected positive values greater than 0. To investigate if the data meets our expectations, we computed for these three variables the percentage of values smaller or equal to 0 (Code Snippet 6).

|  |  |
| --- | --- |
| Field | Values <= 0 (%) |
| fare | 0,14 |
| trip\_seconds | 5,09 |
| trip\_miles | 20,85 |

Table 2 Percent of values smaller or equal to 0.

(6)

SELECT

  COUNTIF(fare <= 0) / count(\*) \* 100  as perc\_zero\_neg\_fare,

  COUNTIF(trip\_seconds <= 0) / count(\*) \* 100 as perc\_zero\_neg\_trip\_seconds,

  COUNTIF(trip\_miles <= 0) / count(\*) \* 100 as perc\_zero\_neg\_trip\_miles

FROM `bt-int-ml-specialization.demo1.taxi\_trips`;

Furthermore, we assumed that trip\_start\_timestamp is smaller than trip\_end\_timestamp. Running the query, we found that 37,79% of the data violated this assumption with the trip\_start\_timestamp being greater or equal to the trip\_end\_timestamp (Code Snippet 7).

(7)

SELECT

  COUNTIF(trip\_end\_timestamp <= trip\_start\_timestamp) / count(\*) \* 100  as perc\_zero\_neg\_time

FROM `bt-int-ml-specialization.demo1.taxi\_trips`;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Fieldname | Count | Mean | Standard Deviation | Minimum | Maximum |
| dropoff\_latitude | 183593808 | 41,90 | 0,04 | 41,65 | 42,02 |
| dropoff\_longitude | 183593808 | -87,66 | 0,06 | -87,91 | -87,53 |
| fare | 211634435 | 13,81 | 45,52 | 0,00 | 9999,99 |
| pickup\_latitude | 187506981 | 41,90 | 0,04 | 41,65 | 42,02 |
| pickup\_longitude | 187506981 | -87,66 | 0,08 | -87,91 | -87,53 |
| trip\_miles | 211652596 | 3,48 | 11,08 | 0,00 | 3460,00 |
| trip\_seconds | 210352969 | 818,94 | 1280,80 | 0,00 | 86400,00 |

Table 3 Descriptive statistics for the three float variables fare, trip\_seconds, trip\_miles, pickup\_latitude, pickup\_longitude, dropoff\_latitude, and dropoff\_longitude.

To get a better understanding for the distribution of the float variables fare, trip\_seconds, trip\_miles, pickup\_latitude, pickup\_longitude, dropoff\_latitude, and dropoff\_longitude we computed descriptive statistics (Table 3) (Code Snippet 8).

(8)

SELECT

  'fare' as field,

  COUNT(fare) AS count,

  AVG(fare) AS mean,

  STDDEV(fare) AS stddev,

  MIN(fare) AS min,

  MAX(fare) AS max

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`

UNION ALL

SELECT

  'trip\_seconds' as field,

  COUNT(trip\_seconds) AS count,

  AVG(trip\_seconds) AS mean,

  STDDEV(trip\_seconds) AS stddev,

  MIN(trip\_seconds) AS min,

  MAX(trip\_seconds) AS max

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`

UNION ALL

SELECT

  'trip\_miles' as field,

  COUNT(trip\_miles) AS count,

  AVG(trip\_miles) AS mean,

  STDDEV(trip\_miles) AS stddev,

  MIN(trip\_miles) AS min,

  MAX(trip\_miles) AS max

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`

UNION ALL

SELECT

  'pickup\_latitude' as field,

  COUNT(pickup\_latitude) AS count,

  AVG(pickup\_latitude) AS mean,

  STDDEV(pickup\_latitude) AS stddev,

  MIN(pickup\_latitude) AS min,

  MAX(pickup\_latitude) AS max

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`

UNION ALL

SELECT

  'pickup\_longitude' as field,

  COUNT(pickup\_longitude) AS count,

  AVG(pickup\_longitude) AS mean,

  STDDEV(pickup\_longitude) AS stddev,

  MIN(pickup\_longitude) AS min,

  MAX(pickup\_longitude) AS max

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`

UNION ALL

SELECT

  'dropoff\_latitude' as field,

  COUNT(dropoff\_latitude) AS count,

  AVG(dropoff\_latitude) AS mean,

  STDDEV(dropoff\_latitude) AS stddev,

  MIN(dropoff\_latitude) AS min,

  MAX(dropoff\_latitude) AS max

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`

UNION ALL

SELECT

  'dropoff\_longitude' as field,

  COUNT(dropoff\_longitude) AS count,

  AVG(dropoff\_longitude) AS mean,

  STDDEV(dropoff\_longitude) AS stddev,

  MIN(dropoff\_longitude) AS min,

  MAX(dropoff\_longitude) AS max

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`

  order by field;

Analogously, we extracted count, first occurrence, last occurrence, and range of days for the timestamp trip\_start\_timestamp (Code Snippet 9).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Count | First Occurrence | Last Occurrence | Range Days |
| trip\_start\_timestamp | 211655493 | 01.01.2013 | 01.01.2024 | 4017 |

Table 4 Descriptive Statistics for trip\_start\_timestamp.

(9)

SELECT

  'trip\_start\_timestamp' as field,

  COUNT(trip\_start\_timestamp) AS count,

  date(MIN(trip\_start\_timestamp)) AS first\_occurrence,

  date(MAX(trip\_start\_timestamp)) AS last\_occurrence,

  TIMESTAMP\_DIFF(MAX(trip\_start\_timestamp), MIN(trip\_start\_timestamp), DAY) AS range\_days

FROM

  `bt-int-ml-specialization.demo1.taxi\_trips`;

The data exploration yielded valuable insights with regards to data preprocessing, feature engineering, and model architecture.

That is, data preprocessing needs to include steps to clean data from inconsistencies such as entries for which trip\_start\_timestamp is greater or equal to trip\_end\_timestamp and methods to cope with missing values. In particular, for the location variables pickup/dropoff\_longitude/latitude we found a significant number of entries with missing values. Furthermore, large standard deviations and value ranges as for instance found for the variable trip\_seconds impose the need for eliminating outliers and applying standardization techniques to normalize the distribution.

Data exploration helped preparing feature engineering in several ways. First, exploring the dataset enabled us to identify our target variable ‘fare’, variables that can serve as predictors such as trip\_miles or trip\_seconds, and variables that can be neglected because they are either not relevant, e.g., payment\_type or constitute redundant information, e.g., pickup\_location. Second, we learnt that temporal information is condensed into two fields: trip\_seconds and trip\_start\_timestamp. trip\_start\_timestamp can be dissolved into several features that might be relevant for the prediction of taxi fares. The timestamp can be split into a month, day, hour, and minute component each of which might yield predictive power. Fares can for instance be smaller at daytime when taxis are competing with public transportation while they might be larger at night when less transportation is available. Besides splitting the timestamps into its different components, one can extract the day of the week. We suggest that fares could be larger on weekends when bars, restaurants, and theatres lure potential customers into the city. Given that we will feed these categorical variables into our model, the feature engineering will need to involve one-hot-encoding.

Lastly, exploring the data also helped to get a first idea of the design of our DNN. DNNs vary in size and can comprise a single hidden layer with few neurons up to hundreds of hidden layers with a multitude of neurons. While large neural networks are associated with complex task such as processing image input where each pixel-channel-combination serves as distinct feature, they are less well suited for simple problems where they risk to overfit to the training data. Given our restricted number of inputs, we will aim for a shallow neural network with few layers. Furthermore, we found that data quality is impaired by missing values and inconsistencies. While we can reduce some of the noise by the preprocessing steps, we can for our network architecture also consider dropout, early stopping and other regularization techniques to prevent the model from overfitting to noisy training data.

1. Feature Engineering

Exploring the data, we determined our target variable to be fare. To predict fare, we focus on 7 variables: trip\_start\_timestamp, trip\_seconds, trip\_miles, pickup\_latitude, pickup\_longitude, dropoff\_latitude, dropoff\_longitude.

To avoid feeding into our model entries with missing values, we first attempted to impute them by filling in the mean of the respective column. However, imputing missing values did not increase our model performance and given the large size of our dataset we decided to simply eliminate rows with missing values. We further restricted the data to consistent entries. That is, we excluded entries where either fare, trip\_seconds, or trip\_miles was smaller or equal to zero. Finally, we discarded entries where trip\_end\_timestamp was smaller or equal to trip\_start\_timestamp. Although this approach eliminated 52.53% of the data, the resulting set still comprised 100.473.417 records and was both cleaner and more consistent (Table: bt-int-ml-specialization.demo1.taxi\_trips\_clean; Code Snippet 10).

(10)

CREATE OR REPLACE TABLE `bt-int-ml-specialization.demo1.taxi\_trips\_clean`AS SELECT

  unique\_key,

  trip\_start\_timestamp,

  fare,

  trip\_seconds,

  trip\_miles,

  pickup\_latitude,

  pickup\_longitude,

  dropoff\_latitude,

  dropoff\_longitude

FROM `bt-int-ml-specialization.demo1.taxi\_trips`

WHERE

  trip\_start\_timestamp is not null

  and fare is not null

  and trip\_seconds is not null

  and trip\_miles 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 fare > 0

  and trip\_seconds > 0

  and trip\_miles > 0

  and trip\_end\_timestamp > trip\_start\_timestamp;

To deal with outliers, we next exported 1 million records and analysed the data in Python. For fare, trip\_seconds, and trip\_miles we plotted histograms. As revealed during data exploration we found that histograms were dominated by outliers (Figure 1A). By visual inspection, we determined a threshold to eliminate outliers. These data were excluded from further processing. We set the threshold to < 80 for fare, < 6000 for trip\_seconds, and < 30 for trip\_miles (Figure 1B; Table `bt-int-ml-specialization.demo1.taxi\_trips\_ex\_outlier`; Code Snippet 11).

(11)

CREATE OR REPLACE TABLE `bt-int-ml-specialization.demo1.taxi\_trips\_ex\_outlier`

AS WITH rank\_cte as(

  SELECT

    unique\_key,

    RANK() OVER (ORDER BY RAND()) as rnd\_rank,

    COUNT(\*) OVER () as total\_rows

    FROM `bt-int-ml-specialization.demo1.taxi\_trips\_clean`

)

  SELECT

    a.trip\_start\_timestamp,

    a.fare,

    a.trip\_seconds,

    a.trip\_miles,

    a.pickup\_latitude,

    a.pickup\_longitude,

    a.dropoff\_latitude,

    a.dropoff\_longitude,

    b.rnd\_rank/b.total\_rows as norm\_rank

  FROM `bt-int-ml-specialization.demo1.taxi\_trips\_clean` a

  INNER JOIN rank\_cte b

  ON a.unique\_key = b.unique\_key

  WHERE a.fare < 80

  AND a.trip\_seconds < 6000

  AND a.trip\_miles < 30

We then standardized data by subtracting the mean and dividing by the standard deviation (Figure 1C).

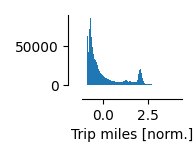
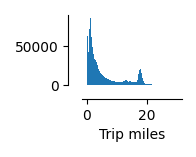
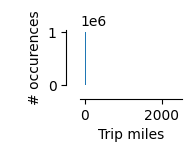


Figure 1 Feature engineering trip\_miles. To determine a threshold for outliers we plotted the distribution of trip\_miles (A). By visual inspection we then set the threshold to eliminate outliers (B). Finally, data was standardized by subtracting the mean and dividing by the standard deviation (C). This procedure was adopted for trip\_seconds and fare.

From the trip\_start\_timestamp we extracted the features day\_of\_week, start\_month, start\_day, start\_hour, and start\_minute applied one-hot-encoding.

To shed light on the importance of our selected features, we fit a DNN. To this end, exported data was split into a training and test set (80%/20%). The data was then used to train a DNN with an input layer, three hidden layers with 64, 32, and 16 units respectively, and an output layer with a single unit for regression output. For the hidden layers we chose a ReLU activation function and for the output layer a linear activation function. We set the learning rate to 0.001, batch size to 32, and the loss function to minimize the mean squared error. Additionally, we applied the ADAM optimizer. To prevent overfitting, we implemented early stopping by halting the algorithm 10 epochs after the minimum validation loss has been reached. Despite its simplicity, the model fit the data well yielding an R² of 0.98.

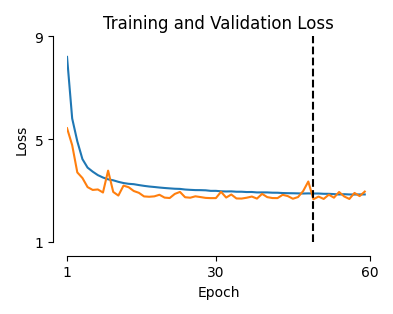


Figure 2 Training and validation loss for a DNN fit to a subset of the training data. Training loss is depicted in blue. Validation loss is depicted in orange. Early stopping led the training to stop at epoch 59. The model’s parameters were then reset to the epoch the with minimum validation loss (2.6584, epoch: 49, dashed black vertical line).

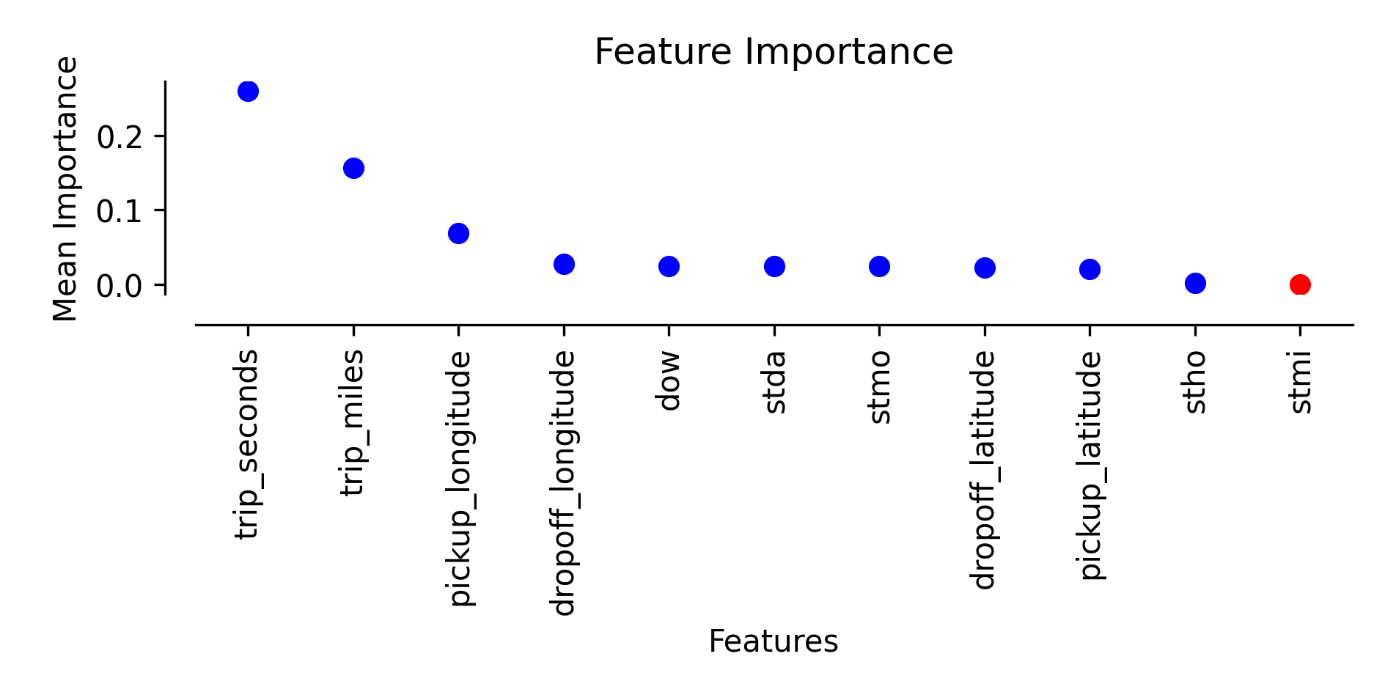
We then determined feature importance by shuffling features in the evaluation dataset, recomputing R² values with this modified dataset and comparing it to the original R² value. The procedure was repeated 100 times for each feature, randomly subsampling 20% of the 1 Mio. records on each trial. For each feature we then computed the mean importance and 95%-confidence-intervals. If 0 was contained in the confidence interval, the feature did not significantly contribute to the prediction and was excluded. Following this rational, we excluded start\_minute where the 95%-confidence interval ranged from -0,0002 to 0,0001 (Figure 3). We then evaluated the retrained model and observed no change in the quality of prediction (R² = 0.98).

Figure 3 Mean feature importance computed by shuffling the respective feature in the evaluation data, recomputing R² and comparing it to the original R² value. stho: start hour, stmo: start month, dow: day of week, stda: start date, stmi: start minute. Points depict mean feature importance. Error bars are small and therefore occluded. Start minute is labelled in red because its mean importance did not significantly differ from 0 (95%-confidence-interval: -0,0002, 0,0001).

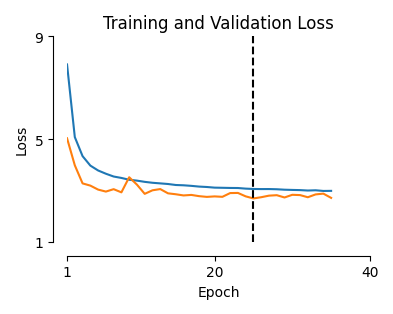


Figure 4 Training and validation loss for a DNN fit to a subset of the training data after excluding the feature start\_minute. Training loss is depicted in blue. Validation loss is depicted in orange. Early stopping led the training to stop at epoch 35. The model’s parameters were then reset to the epoch the with minimum validation loss (2.6897, epoch: 25, dashed black vertical line).

1. Preprocessing and the data pipeline

The core of our preprocessing pipeline consisted of a dockerized apache beam pipeline that was executed by a Dataflow flow job for online prediction. While the preprocessing steps did not differ between training and online serving, the pipelines were adjusted to fit the specific purpose.

Training preprocessing started in BigQuery where we derived a clean table from our initial dataset. To this end, we set excluded outliers and incomplete data as explained in the data exploration section. In brief, we excluded rows where any of the predictor or the target were null. We discarded entries where either fare, trip\_seconds or trip\_miles were not greater than 0 and entries for which the trip\_end\_timestamp was not greater than the trip\_start\_timestamp. Furthermore, we applied thresholds as determined earlier and excluded rows with extremely large fares or long trips as derived from trip\_seconds or trip\_miles.

This table was then queried by our main\_batch.py function. The function is started manually by the user and triggers an apache beam pipeline which consists of five steps:

1. The data is read from the BigQuery Table.
2. The custom written function utils.add\_date\_info\_fn performs feature extraction by computing the day\_of\_week, start\_month, start\_date, and start\_hour from the trip\_start\_timestamp, which is deleted thereafter.
3. A transform function applies one hot encoding to all categorical variables and z-standardizes all float variables. This transform process yields results that are relevant for preprocessing data for later online prediction such as the mean and the standard deviation of float variables. To make this information available, we stored the transform step as artifact in a bucket.
4. Preprocessed data were then converted to tf.train.Example objects and stored in the cloud bucket as .tfrecord file.

For online serving, we assume that the model will only be queried with complete input data. Thus, we do not need to consider cases in which data is partially missing. While we excluded outliers for training, the deployed model can be called with outliers. Although its predictions might not be as accurate as for standard data, we argue that it will still provide a rough benchmark for determining the fare. Furthermore, outliers represent edge cases that occure seldomly. Hence, inaccurate predictions will not affect the success of the taxi company.

Preprocessing data for online serving required a slightly different pipeline that we implemented in main\_inference.py. It consists of seven steps:

1. The apache beam pipeline reads messages from a subscription associated with a source topic on pubsub.
2. Incoming messages are then transformed to python dictionaries.
3. A window function binned data in one second chunks.
4. In step 4 and step 5 data are preprocessed analogue to the procedure applied before training. That is, time variables are computed from the start\_trip\_timestamp and scaling and one-hot-encoding are applied to numerical and categorical variables respectively.
5. Preprocessed data is then served to the model’s endpoint in order to obtain the fare prediction.
6. Lastly, the predicition and the model’s input is messaged to a sink topic on pubsub.

The main\_inference.py file is dockerized and the image stored in the container registry. From here, it is executed by dataflow job.

1. Machine learning model design and selection

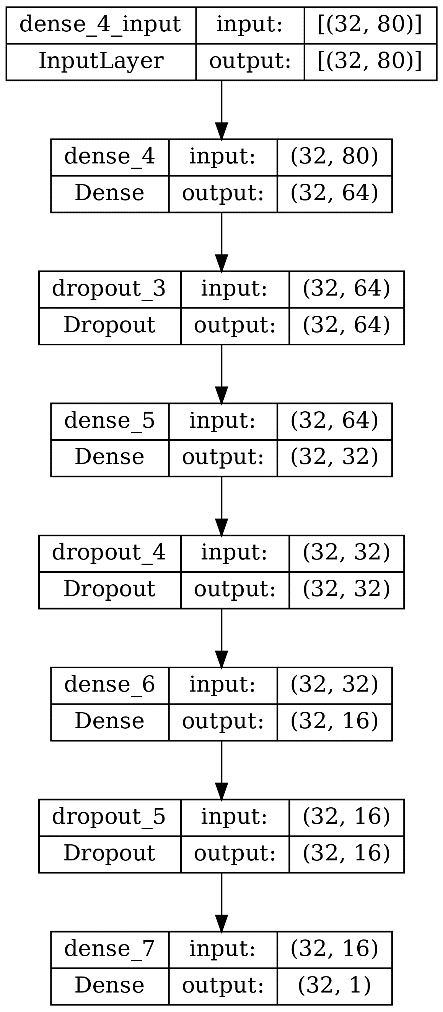
In this white paper, we describe our approach to developing a Deep Neural Network (DNN) for predicting taxi trip fares using features such as trip duration, distance, and the day of the week. The model is built with TensorFlow to ensure compatibility with TensorFlow-based systems.

Figure 5 Model architecture. The model consists of an input layer, three hidden layers and an output layer. The shape of the data fed to the model’s input layer is determined by the batch size (32) and the number of features (80). The output of the last layer is a single node reflecting the predicted taxi fare.

Our DNN consists of an input layer, several hidden layers, and an output layer designed for regression tasks. The network starts with 64 neurons in the input layer and uses the ReLU activation function. To prevent overfitting, we applied dropout after each layer, a decision based on optimizing model performance. The model's structure includes hidden layers with decreasing neuron numbers (32 and 16), leading to a single-neuron output layer with a linear activation function for fare prediction. This architecture helps compress the input data into a meaningful representation for predicting fares.

We used the Adam optimizer and aimed to minimize the Mean Squared Error (MSE). The choices for the learning rate and dropout rates were optimized through hyperparameter tuning with Vertex AI, ensuring that the model operates with the most effective parameters. To avoid overtraining, we incorporated an early stopping mechanism that stops training if the validation loss doesn't improve after ten epochs, reverting to the best set of weights found during training. The model was trained on an 80/20 split of the data for a maximum of 100 epochs with a batch size of 32. Its effectiveness is highlighted by achieving an R² value of 0.98 on a data subset, indicating excellent predictive accuracy.

In summary, through careful architecture design, optimal hyperparameter selection via Vertex AI, and strategies like dropout and early stopping, we developed a DNN model that accurately predicts taxi fares. This work demonstrates the practical application of machine learning in analyzing and predicting based on complex data.

1. Machine learning model training and development

For model training, we restricted data to 5 Mio. records. On the one hand, this assured that model training completed within a reasonable time window, on the other hand we could assure that the dataset size is large enough to train a model with high validity. 80% of these 5 Mio. records were used for training (taxi\_trips\_train table) and 20% were stored in a separated table for model evaluation (taxi\_trips\_eval table). The assignment of records to train or evaluation datset was random. To train our model in Vertex AI, we developed a Kubeflow Pipeline. The pipeline consists of the following steps:

1. We designed a Dataflow job to batch preprocess training data. The batch preprocessing itself, was implemented in an Apache Beam pipeline which first reads the data from a Big Query table, extracts temporal information on the trip\_start\_timestamp and transforms the data by employing one-hot-encoding and feature scaling. Lastly, the data is written as .tf\_record file to a cloud bucket. To ensure that data for model evaluation and data that is later served online to the model are processed in the same manner, we saved the transformation steps as artifact to a cloud bucket.
2. The pre-processed data were then used for hyperparameter tuning. Hyperparameter tuning involved two parameters: learning rate and dropout rate. Learning rate values were sampled on a logscale ranging between 0.001 and 1. Dropout rate was sampled on a linear scale ranging between 0.05 and 0.3. We tested 5 combinations in parallel. The model training aimed to minimize the validation loss defined as the mean squared error. The mean squared error is a good choice for evaluating a regression model because it provides a concrete quantitative measure of how close the model's predictions are to the actual data. By squaring the errors and taking the average, MSE penalizes larger errors more severely, thus emphasizing accuracy and making it particularly useful when large errors are undesirable in predictions. To assess the model’s performance, we split off 20% from the training dataset and used it for testing. By implementing dropout layers and incorporating an early-stopping mechanism, we assured that the model would not be overfit to the training data.
3. The model with the best performance, i.e., lowest mean squared error, was uploaded to the model registry, and deployed to an endpoint for online serving.
4. A second Data Flow job was started. The Apache Beam pipeline of this second job was set up to read incoming messages from a source Pub/Sub topic, preprocess these information relying on the same steps as preprocessing for model training, sending the preprocessed data to the model’s endpoint and writing the result back to the sink Pub/Sub topic.
5. To evaluate the performance of the model on unseen data, we integrated an evaluation step which is outlined in the next section.
6. Machine learning model evaluation:

To assess the performance of the selected model, we initially reserved ~1 Mio. records for evaluation. A Data Flow job incorporated in our Kubeflow pipeline processed the data analogously to the training data. The data was then fed to the deployed model for batch prediction. In a model-evaluation-regression block, we assessed the key performance metrics and found that the model consistently achieved an R²-Score well above 0.9. The R-squared statistic is beneficial for evaluating a regression model as it provides a measure of how well the predicted values fit the actual data, expressed as a proportion of total variation in the dataset. A higher R-squared value (closer to 1) indicates a higher proportion of variance in the dependent variable that can be explained by the independent variables in the model, making it very useful to assess the model's effectiveness in explaining the variance of the data.