MTH786P Machine Learning with Python: Final Project

Flight price prediction (Regression)

Santosh Sullipalayam Rajasekaran
Student ID: 250825222

January 2026

Abstract

This project studies the Flight Price dataset with the aim of predicting airline ticket prices by exploring regression models. The dataset presents a realistic problem, as it combines categorical, time-based, and numerical features that require careful preprocessing. A key part of the work focuses on data cleaning and feature engineering, including converting string-based variables into meaningful numerical features using ordinal and one-hot encoding.

All models are implemented using NumPy in order to better understand the underlying learning algorithms. Several regression methods are explored, including linear models, regularised regression, polynomial regression, kernel-based methods, and a simple neural network. Model selection and hyperparameter tuning are carried out using 10-fold cross-validation, and final performance is evaluated on a held-out test set. The main evaluation metric is the coefficient of determination (R^2), with mean squared error (MSE) also reported.

The results indicate that the representation of the features and the models used have a great influence on model performance, and the best results for this dataset are obtained by using nonlinear models, particularly Kernel Ridge Regression with the RBF kernel.

1 Introduction and problem statement

The goal of this project is to predict the flight ticket price from the available flight information. The dataset contains categorical, time-based, and numeric fields, so the main challenge is turning real-world flight characteristic descriptions into a clean numerical design matrix that models can learn from.

I found this dataset very interesting because small feature planning decisions (such as how time, route, and stops are represented) were key to model behaviour. Many features were initially stored as strings, so I had to design logical methods and encodings to convert them to numeric values (see Section 3.2). This made the project feel more like an experiment in understanding models, and extracting as much information as possible from the given data rather than optimising for a fit.

Dataset choice. I selected the Flight Price dataset also because it is representative of a real-world regression problem where data cleaning, representation choices, and feature design play a central role. Beyond just predicting prices, the dataset allowed me to introduce my own ideas (or *a priori* in some terms). The task is formulated as a supervised regression problem with ticket price as the continuous target variable.

2 Exploratory Data Analysis

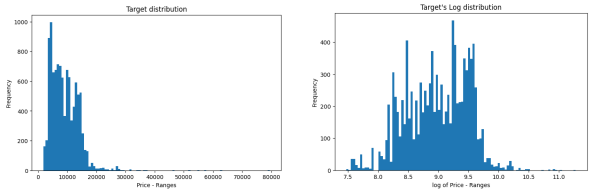
2.1 Data cleaning

The dataset contained very few missing values and duplicate entries.

All rows with missing values were removed. Duplicates required more careful treatment; duplicated rows might not always be entry errors. In particular, some entries might have differed in seat number or even had the same plane setup. Since the prediction task concerns ticket prices rather than individual seats, records sharing the same ticket were considered duplicates and only one instance was retained each time.

2.2 Target distribution

The target price has a strong right tail, due to a smaller number of costly tickets, as shown in Figures 1a and 1b. To assess whether a log-transform manages the influence of the high-price tail and improves linear modelling, I also trained linear models on $\log(Y)$ and mapped predictions back to the original scale when needed (de-log transform: exponentiate).



(a) Raw target price Y. (b) Log-target $\log(Y)$.

Figure 1: Target distributions (raw and log-transformed).

2.3 Correlations and quick feature checks

Instead of just looking at correlations in a single final matrix of data points, I also computed correlations *within feature blocks* (numeric, one-hot, route, etc.). Although this was extra work.

One observation was that the column `NDC` exhibited a visible correlation with the airline feature `Trujet`. Although correlated, `Trujet` had clearer meaning and stronger explanatory value. To reduce redundancy without losing information, `NDC` was therefore removed from the final feature set.

Table 1: Top absolute correlations with the raw target price Y.

Feature	Corr. with Y
Stops count	0.5992
BOM	0.5768
DEL	0.5281
Duration (min)	0.4726
Air India	0.3676

Table 2: Top absolute correlations with the log-target $\log(Y)$.

Feature	Corr. with $\log(Y)$
Stops count	0.6460
BOM	0.5795
DEL	0.5251
Duration (min)	0.4763
Air India	0.3796

2.4 Full feature correlation map

I plotted the complete correlation map of the feature matrix X (Figure 2), which acted as a quick check for strong redundancies.

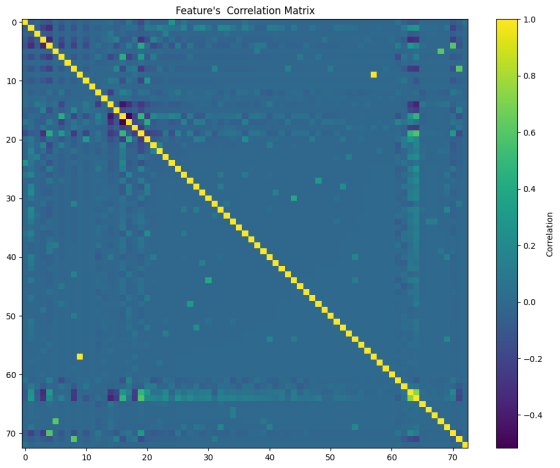


Figure 2: Full correlation heatmap of the feature matrix X.

2.5 Duration vs price

One of the most rational relationships in the dataset would be between flight duration and

price (at least for me). As shown in Figure 3, longer flights tend to cost more, but unlike expected the relationship is not perfectly linear, likely due to factors such as layovers and routing.

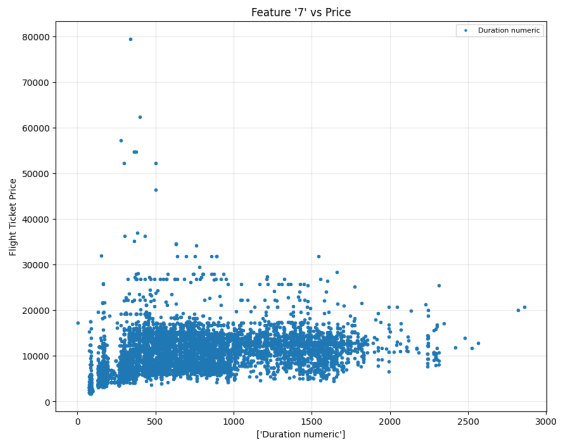


Figure 3: Flight duration (minutes) vs ticket price.

This plot suggests the possibility of nonlinear models such as polynomial regression and kernel ridge regression suiting the data better than linear models.

2.6 Correlation of target with all features

To summarise feature impact in a simple way and later compare with the weights of good models, I plotted the correlation between the target price Y and features (Figure 4). This plot shows a possible high usage for some features but cannot be taken at face value.

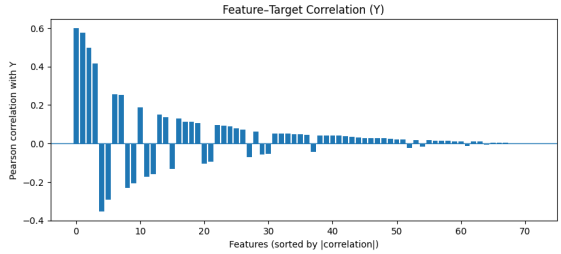


Figure 4: Correlation of flight price Y with all features in the design matrix.

3 Methods

3.1 Important implementation rule

As required in the assessment, I implemented the algorithms completely using NumPy only. I still used some Pandas and plotting libraries for exploration and visualisation.

3.2 Feature engineering strategy

A single type of encoder was not applied to the whole dataset. Instead, each feature type was treated uniquely:

- **Ordinal features:** *e.g. total stops* was kept as an integer (ordering matters).
- **Nominal categorical features:** e.g. Airline, Source, Destination, and similar fields were one-hot encoded.
- **Time features:** e.g. dates and times were converted to numerical forms (minutes since midnight, ordinal day index, simple weekend indicator).
- **Route features:** I extracted intermediate cities and kept only the most frequent 100 to avoid a huge sparse matrix.
- **Dual-target view:** I evaluated both Y and $\log(Y)$ (then mapped predictions back when needed) to understand outliers and relative error.

Inductive bias via features. Some feature choices (e.g. adding a weekday/weekend marker) can be seen as injecting a simple prior: it gives the model a structured hint about patterns it might want to fit, without changing the model class.

Encoding choices. All categorical features were transformed into numeric features using a combination of ordinal encoding and one-hot encoding mainly. Ordinal encoding was used only when an order was present (e.g. number of stops), while purely categorical variables were one-hot encoded. This avoids imposing artificial ordering while keeping the design matrix meaningful.

3.3 Models implemented

I coded and compared the following models (all trained using NumPy code):

- OLS (closed-form solution where appropriate, and compared normalisation methods),
- Ridge regression (L2 regularisation, tuned λ),
- Lasso-style approach (L1 effect using the optimisation routine in my notebook),
- Polynomial regression (tested degrees as a controlled nonlinearity),
- Kernel Ridge Regression with RBF kernel (tuned γ and λ),
- A small Neural Network (1 hidden layer, tanh activation, gradient descent).

3.4 Evaluation protocol and metrics

I used:

- **10-fold cross-validation** for model selection and stability,
- a **held-out test set** for final reporting,
- R^2 **and MSE** as the main metrics.

The primary metric is R^2, as it provides an intuitive, scale-free and context-free numeric score and is easier to compare visually across models. MSE is also reported, but was not used much because its absolute value requires contextual interpretation due to the scale of ticket prices.

3.5 Normalisation strategies

Several scaling strategies were tested during model development: standardisation, robust scaling, logarithmic transformation, and min–max normalisation to $[0, 1]$. For OLS, standardisation consistently produced the most stable cross-validation performance. Once established, this normalisation choice was fixed and applied uniformly to all subsequent models to ensure fair comparison.

3.6 Training procedure

All models were evaluated using 10-fold cross-validation. For linear models (OLS, Ridge, Lasso), both Y and $\log(Y)$ targets were tested. Polynomial regression treated the polynomial degree as a hyperparameter. Ridge and Lasso used the regularisation parameter λ as the tuning variable; for Ridge, two tuning rounds were used to optimise its search, but performance remained limited by its linearity.

For Kernel Ridge Regression with an RBF kernel, the hyperparameters (λ, γ) were tuned jointly using 10-fold cross-validation. The $\log(Y)$ target was not used in this case, as combining logarithmic scaling with an RBF kernel can amplify extreme values and destabilise training. Neural networks were also evaluated using 10-fold validation with a fixed simple architecture (one hidden layer).

3.7 Train, validation, and test strategy

Hyperparameters were tuned on the training and validation folds. Models were first compared based on validation (cross-validation) performance, and only the best-performing parameters were evaluated on the held-out test set. This separation provides a good estimate of generalisation performance.

4 Results

4.1 RBF Kernel Ridge hyperparameter search

For the RBF Kernel Ridge model, a grid search over (λ, γ) was carried out using 10-fold cross-validation. While reporting the single best pair, the full performance surface over the grid was visualised (Figure 5). Luckily I was able to find the maximum score coordinate in the hyperparameter grid.

4.2 Cross-validation comparison

Cross-validation performance was compared across models and their respective hyperparameter grids. For each model, the best

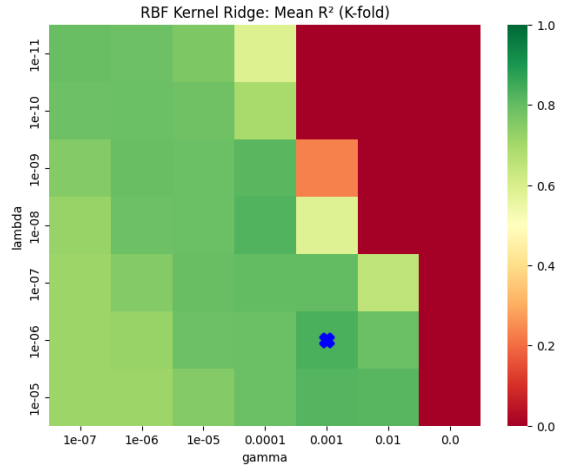


Figure 5: Cross-validation performance over the (λ, γ) grid for RBF Kernel Ridge Regression.

parameters (selected by best mean R^2 over the hyperparameter search) was retained, and the resulting mean $\pm$ standard deviation across the 10 folds is reported in Table 3. Overall, the RBF Kernel Ridge model achieves the highest average performance in CV and will be shown to perform well in the test set.

4.3 Lasso regularisation behaviour

To understand the effect of the regularisation strength λ in Lasso regression, performance was evaluated across a wide range of values and summarised in Figure 6. The plot below illustrates the expected bias–variance trade-off: very small λ values lead to weak convergence and risk overfitting, while very large λ values over-penalise the weights and underfit.

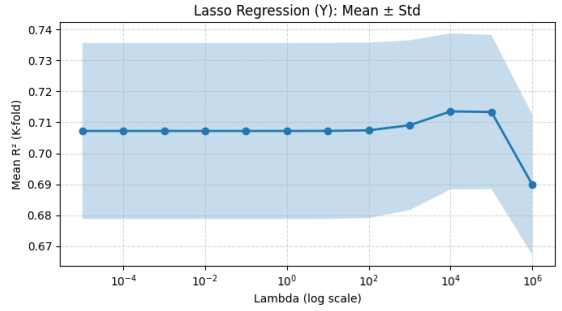


Figure 6: Lasso regression performance as a function of the regularisation parameter λ.

4.4 Test-set comparison

On the test set, the ranking stays consistent, indicating a good fit across the models: RBF-KRR is best, then Polynomial, then linear models (Table 4).

4.5 RBF-KRR: true vs predicted (sorted by Y)

To further understand how the best-performing model behaves across the price range, the test prices and the RBF Kernel Ridge Regression predictions are plotted after sorting samples by the true target Y (Figure 7). This sorting step only reorders the samples; it does not transform Y. Plotting predictions in the same order as the sorted ground truth makes oscillations or errors easier to see, especially in the high-price tail.

5 Implementation

This section highlights a few implementation choices that may look unconventional at first glance, but were useful for debugging, interpretation, and fair comparison between models.

5.1 Feature blocks instead of a single pipeline

Instead of transforming everything in one chained pipeline, separate feature blocks were built (time block, route block, one-hot block, numeric block). Although this required more implementation effort, it made it easy to:

- check distributions and correlations per block,
- debug mistakes (especially route/time parsing),
- explain exactly what each group contributes.

5.2 Dual-target view: Y and $\log(Y)$

It was notable how much the target changes the behaviour of the problem. Training on $\log(Y)$ often reduced the impact of very expensive

Table 3: Cross-validation performance (mean ± std across 10 folds).

Model	Target	Best config	R2_mean	R2_std	MSE_mean	MSE_std
RBF-KRR	Y	lambda=1.000e-06, gamma=1.000e-03	0.8337	0.0340	3,597,004	790,903
POLY	Y	degree=4	0.7950	0.0225	4,446,811	692,346
OLS	Y	normaliser=LOGX	0.7504	0.0472	5,415,473	1,200,332
LASSO	Y	lambda=1.000e+04	0.7135	0.0252	6,197,089	741,983
RIDGE	Y	lambda=4.440e+01	0.7135	0.0251	6,197,665	745,049
NN	Y	hidden=1028, lr=1e-3, iters=500, tanh	0.6487	0.0415	7,687,032	1,627,442
OLS	logY → Y	normaliser=LOGX	0.7627	0.0344	5,136,930	871,165
POLY	logY → Y	degree=2	0.7355	0.0245	5,734,698	809,183
LASSO	logY → Y	lambda=1.000e+01	0.7035	0.0240	6,418,887	776,810
RIDGE	logY → Y	lambda=2.000e-03	0.7014	0.0289	6,474,902	948,328
NN	logY → Y	hidden=1028, lr=1e-3, iters=500, tanh	0.6143	0.0459	8,443,402	1,802,671

Table 4: Test-set performance for the best configuration of each model.

Model	Target	Best config	R2_test	MSE_test
RBF-KRR	Y	lambda=1.000e-06, gamma=1.000e-03	0.8433	3,055,854
POLY	Y	degree=4	0.7960	3,978,623
OLS	Y	normaliser=LOGX	0.7635	4,614,041
RIDGE	Y	lambda=4.440e+01	0.7054	5,746,266
LASSO	Y	lambda=1.000e+04	0.7050	5,754,065
NN	Y	hidden=1028, lr=1e-3, iters=500, tanh	0.6509	7,411,210

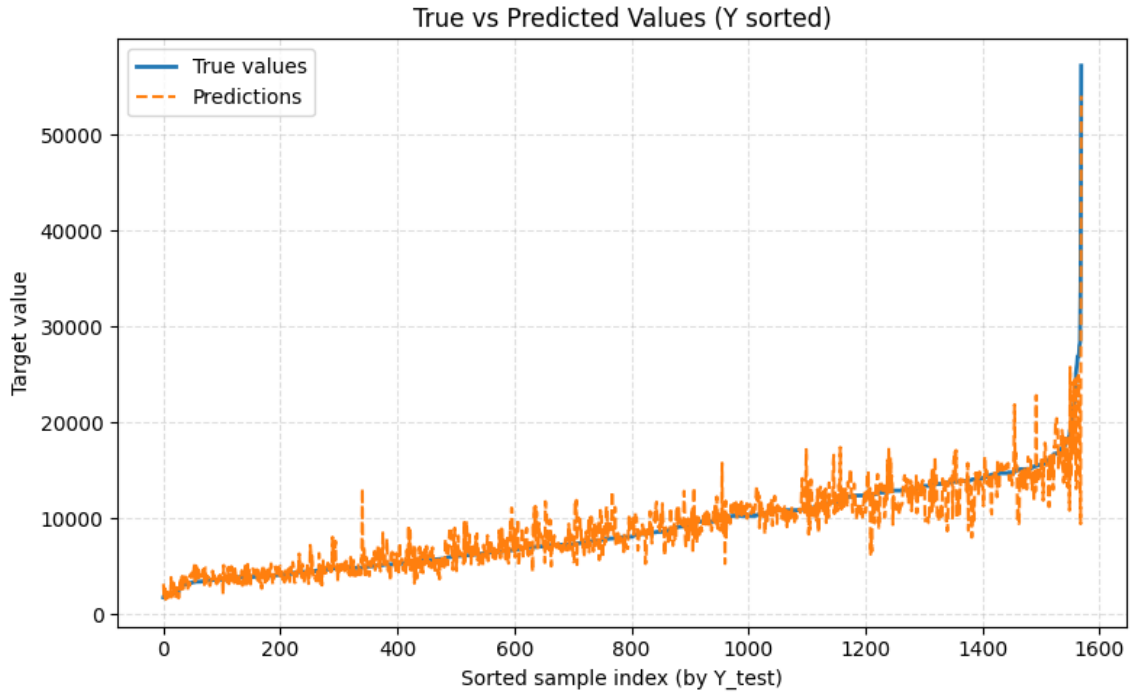


Figure 7: RBF-KRR on the test set: true vs predicted values, with samples sorted by the true target Y.

flights, but sometimes the raw target gave better absolute MSE. Keeping both targets provided a practical way to understand errors.

5.3 Consistent train/test split for Y and $\log(Y)$

A small but important detail is that the train/test split was created only one way and then reused for both target choices. The split indices were also sanity-checked to match. This prevents accidental unfair comparison between Y and $\log(Y)$ runs.

5.4 Uniform tuning framework

All models were tuned using the same cross-validation basis and stored in a single results structure. This made the final comparison table consistent and avoided "special treatment" for any model.

5.5 Neural network baseline

The neural network implementation was left to be simple (one hidden layer, tanh, gradient descent). It did not beat the best kernel model here, but:

- it confirmed that nonlinearity helps (as seen in the improvement),
- it highlighted sensitivity to learning rate and iteration count,
- it provided a clean baseline for "hand-built" deep learning in NumPy.

6 Conclusions

Model behaviour insights. Linear models (OLS, Ridge, Lasso) consistently assigned larger coefficients to duration, number of stops, and major hub indicators, reflecting intuitive pricing drivers. In contrast, the RBF Kernel model relied on a more distributed set of features, capturing nonlinear interactions that linear weights cannot represent explicitly. These differences highlight how model structure affects feature usage and interpretability.

This project made me understand the idea that the largest performance gains usually come from representing the data right, rather than from pushing model complexity alone. My best model was RBF Kernel Ridge Regression, which achieved the strongest R^2 on both cross-validation and the test set.

Correlation, linear weights, and nonlinear feature importance. Correlation analysis with the target price Y tells us that the number of stops is its most correlated feature, indicating that flights with more stops tend to be more expensive on average. However, correlation alone does not indicate the direction or structure of the relationship once multiple features interact.

Inspection of the OLS coefficients gives a clearer linear interpretation: premium travel indicators and certain routes are associated with large positive weights, corresponding to upward shifts in predicted prices, whereas several economy-focused carriers have large negative weights. Although the number of stops is highly correlated with Y, its linear coefficient is comparatively smaller. This phenomenon is shared with correlated variables such as duration, route, and airline type.

Table 5: Largest-magnitude coefficients from the OLS model (standardised features).

Feature	Weight	Effect on Y
Jet Airways Business	+1078.18	Strong increase
Business class	+673.14	Increase
JLR	+533.09	Increase
IXU	+429.49	Increase
DED	+390.86	Increase
GoAir	−959.24	Strong decrease
SpiceJet	−956.13	Strong decrease
Air Asia	−942.42	Strong decrease
Trujet	−940.90	Strong decrease
IndiGo	−928.38	Strong decrease

For the RBF Kernel model, permutation-based feature importance (Table 6) highlights a different structure. Luxury and major city indicators, and continuous time-related features dominate, while the number of stops remains relevant but is no longer the primary driver. This reflects the nonlinear nature of the model, where the influence of stops depends on interactions with timing, route structure, and airline characteristics, not just acting as an independent linear factor.

Table 6: Top permutation-based feature importances for the RBF Kernel model. Larger ΔMSE implies higher importance.

Feature	ΔMSE
Date (numeric)	2.60×10^7
Banglore	2.56×10^7
Duration (numeric)	1.71×10^7
Delhi	1.14×10^7
Kolkata	5.99×10^6
Jet Airways	4.99×10^6
BOM	4.72×10^6
Arrival time (numeric)	4.50×10^6
Stops count	4.44×10^6

Overall, these results suggest that correlation might be useful for initial exploration, but model-based importance provides an accurate picture of how features contribute to predictions, particularly for nonlinear models.

Limitations. A limitation of the linear models (OLS, Ridge, Lasso) is that they can only capture relationships that are (approximately) linear in the chosen features. Conversely, the RBF kernel model imposes a strong *radial* similarity structure in feature space. While this flexibility improved performance, it may still miss patterns that are not well described by either linearity or radial distance.

Future research. If I had more time, I would push further on:

- dimensionality reduction / feature compression (e.g. SVD or PCA) to improve input data value per column,
- better route representations (still compact, but richer),
- more careful outlier handling for very expensive flights,
- small ensembles (still keeping the implementation simple and clear).

References

[1] N. Perra, *MTH786P: Machine Learning with Python – Lecture Notes and Lab Materials*, Queen Mary University of London, 2025–2026.

[2] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*, Cambridge University Press, 2020.

[3] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*, 2nd ed., Springer, 2021.

[4] OpenAI, *ChatGPT*, used for clarifying programming concepts, debugging assistance and sanity checks. `https://chat.openai.com`

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, *Language Models are Unsupervised Multitask Learners*, OpenAI, 2019.