Contents

All figures are representing the 1st-99th percentile, to deal with outliers and to help with clarity in seeing trends.

For fitting, all entries in the dataset have been used, however, for scatter graphs, the number of displayed entries has been extremely limited (less than 1% in some cases) to allow for better visual representation.

Data / Domain Understanding and Exploration

Meaning and Type of Features; Analysis of Distributions

```
# 1st and 99th percentiles (raw prices)
p01, p99 = prices.quantile([lower_bound, upper_bound])

plt.figure(figsize=(7,4))
plt.hist(prices, bins=60, range=(p01, p99))
plt.xlabel("Price (£)")
plt.ylabel("Count")
plt.title("Price distribution (1st-99th percentile, raw)")
plt.show()
```

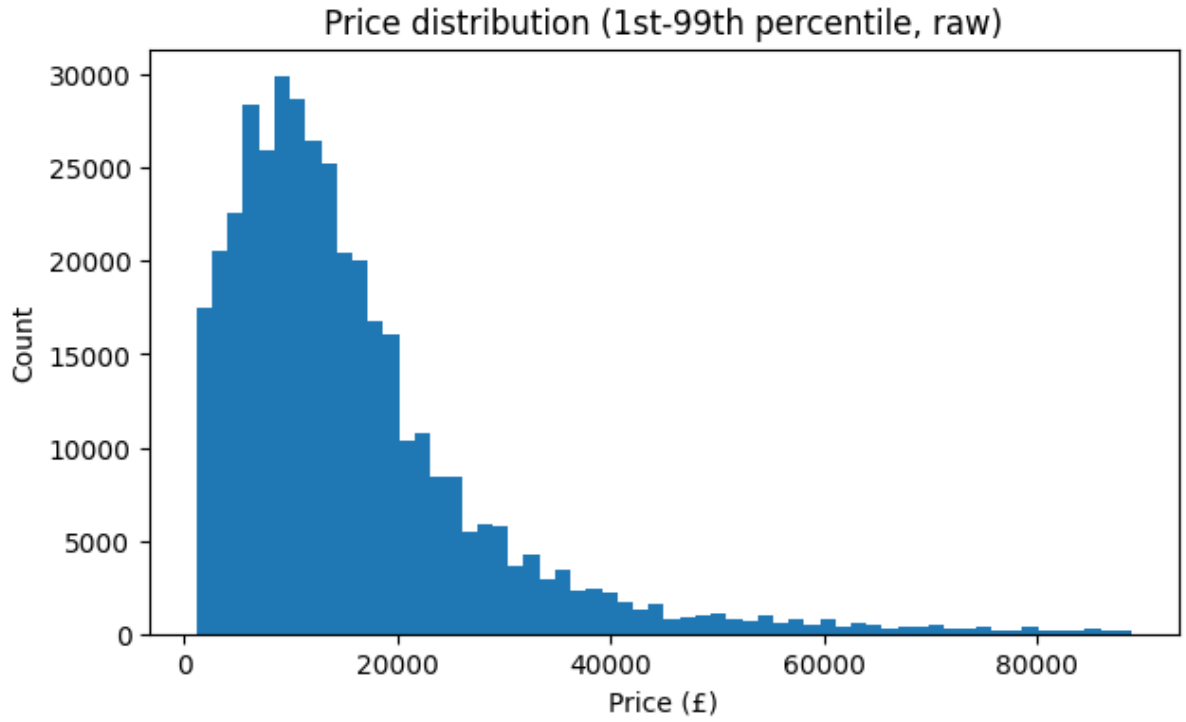

Figure 1: Histogram of car prices plotted against the count

The target variable, price, exhibits a heavily right-skewed distribution. Most vehicles are concentrated at lower price points, and a small number of high-value cars forming a tail.

This shape is typical of a used-car market and indicates that these extreme values may disproportionately influence model fitting if left untreated.

This skewness suggests that a transformation of price would be beneficial for fitting the regression models.

```
# Log1p prices
log_prices = np.log1p(prices)
log_p01, log_p99 = log_prices.quantile([lower_bound, upper_bound])

plt.figure(figsize=(7,4))
plt.hist(log_prices, bins=60, range=(log_p01, log_p99))
plt.xlabel("log1p(Price)")
```

```
plt.ylabel("Count")
plt.title("Price distribution (1st-99th) percentile, log1p)")
plt.show()
```

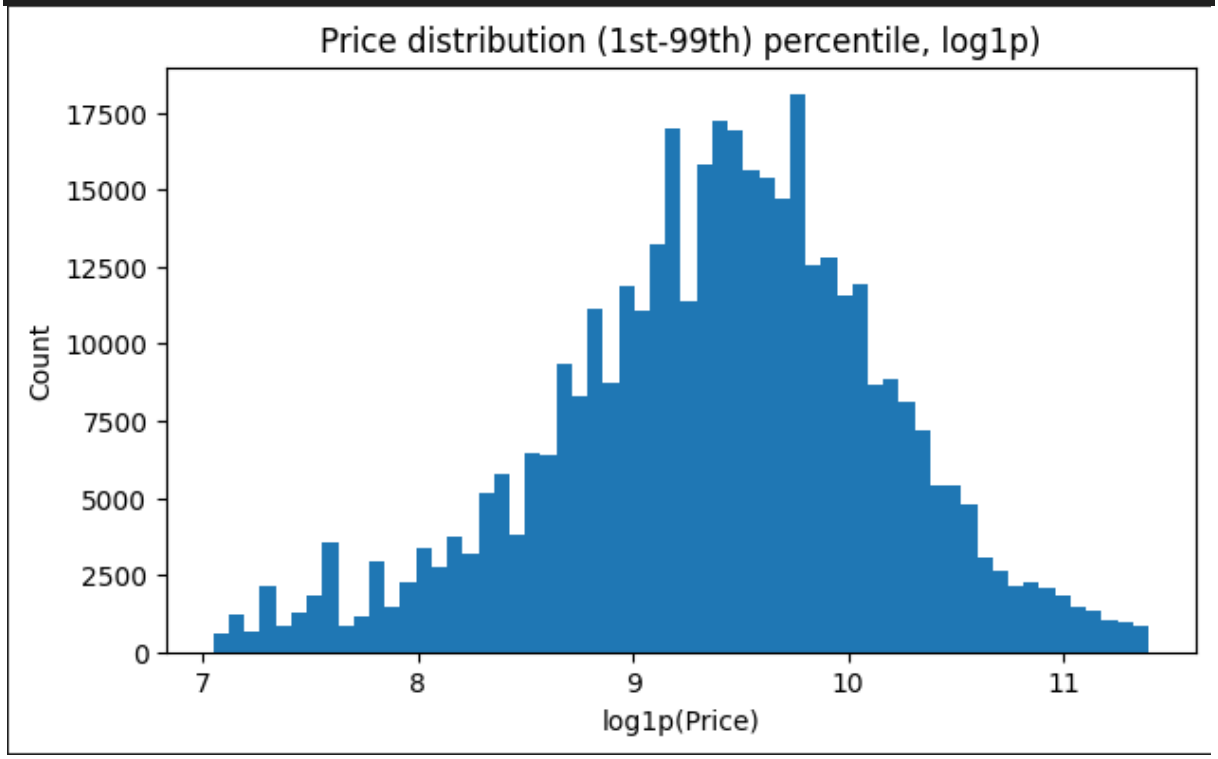

Figure 2: Histogram of car prices (log-transformed) against the count

Applying a log(1+price) transformation has significantly reduces the skewedness and compressed the influence of high value cars. The result is a distribution that is more symmetric and closer to normality, which is more suitable for linear regression and helps stabilise variance.

This transformation was used in the later modelling stages.

```
# Numeric feature distributions (using predefined percentile bounds)
for col in ["mileage", "year_of_registration"]:
    if col in adverts.columns:
        data = adverts[col].dropna()

        # Use pre-defined percentile bounds
        p_low, p_high = data.quantile([lower_bound, upper_bound])

        plt.figure(figsize=(7,4))
        plt.hist(data, bins=60, range=(p_low, p_high))
        plt.xlabel(col)
        plt.ylabel("Count")
        plt.title(
            f"Distribution of {col} (1st-99th percentile)")

        # Force integer ticks for year_of_registration
        if col == "year_of_registration":
            plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
```

```
plt.show()
```

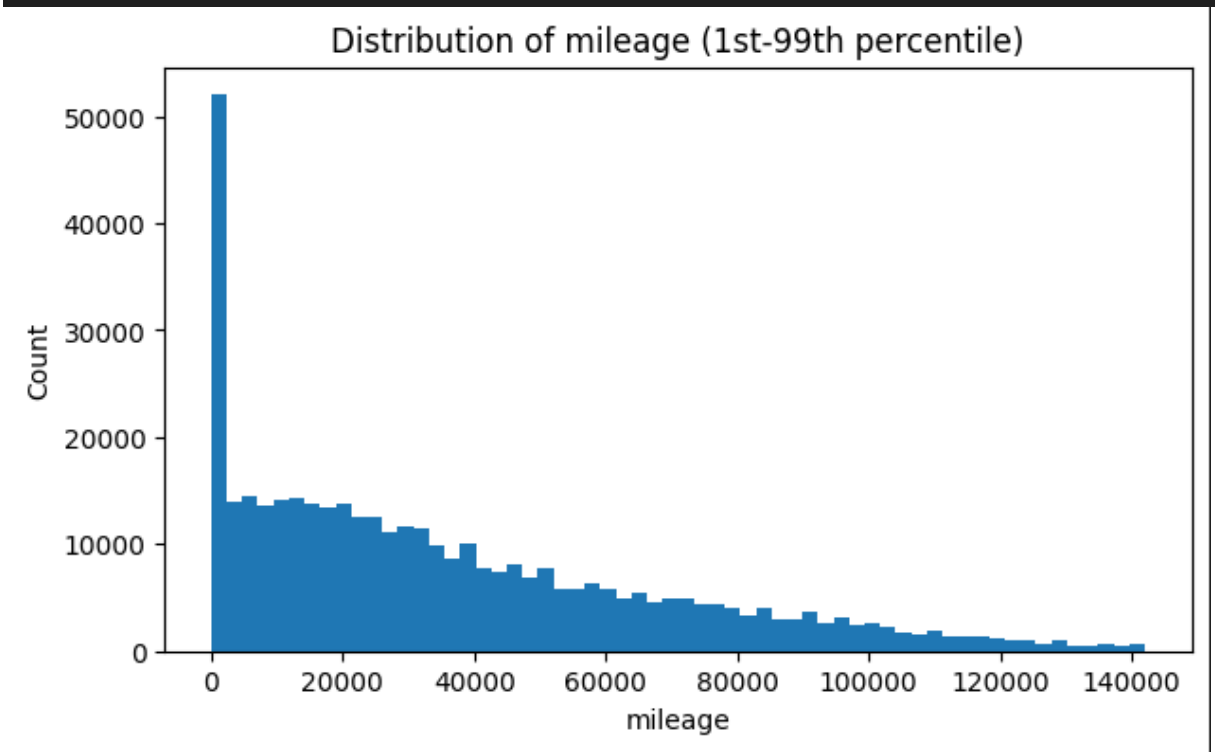

Figure 3: Mileage against count

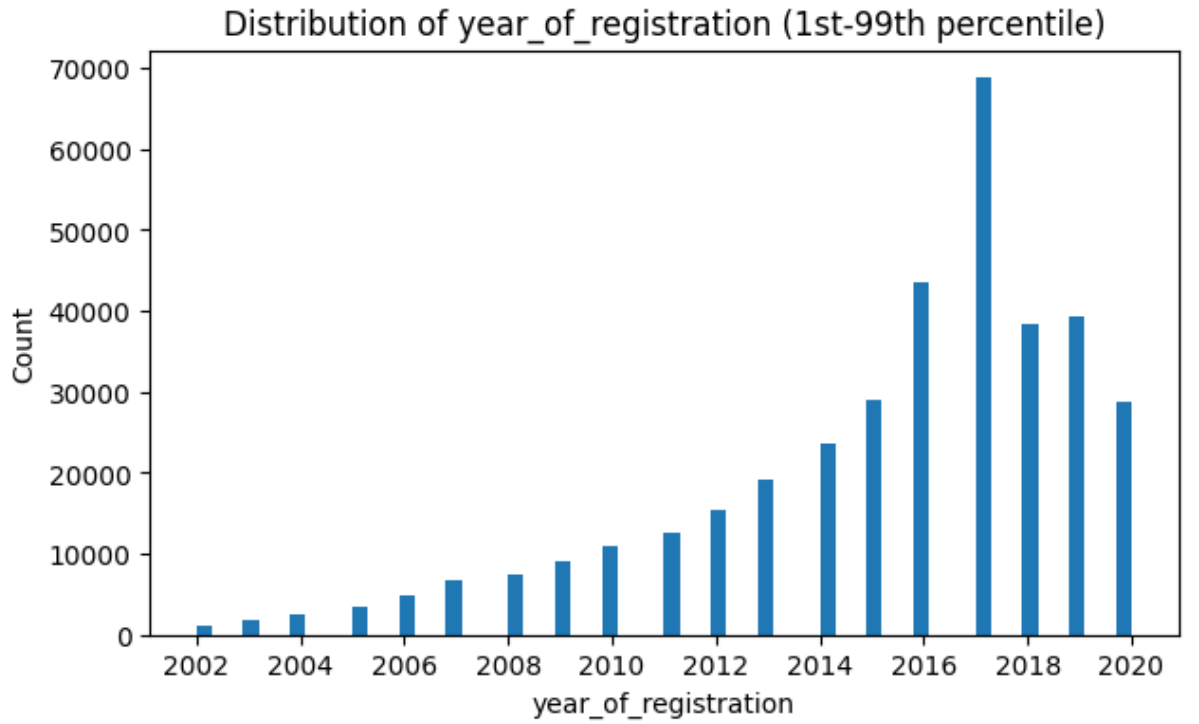

Figure 4: Reg. Year against count

As seen in Figure 3, mileage is a numerical feature representing the vehicle usage (distance travelled in miles) and shows a right-skewed distribution. This shows us that most cars have relatively moderate mileage, especially new cars which are practically guaranteed to have 0 mileage. Very few

cars have very high usage. As mileage is measured in a different scale from other numerical features, there will be a need for feature scaling prior to applying distance-based models such as K-Nearest-Neighbour. Without this scaling, KNN fitting will be dominated by mileage.

```
cleaned_ads.info()
<class 'pandas.core.frame.DataFrame'>
Index: 401674 entries, 0 to 402004
Data columns (total 11 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   mileage               401549 non-null  float64
 1   reg_code              369863 non-null  object 
 2   standard_colour       396330 non-null  object 
 3   standard_make         401674 non-null  object 
 4   standard_model        401674 non-null  object 
 5   vehicle_condition     401674 non-null  object 
 6   year_of_registration  368363 non-null  float64
 7   price                 401674 non-null  int64  
 8   body_type             400866 non-null  object 
 9   crossover_car_and_van 401674 non-null  bool   
 10  fuel_type             401096 non-null  object 
dtypes: bool(1), float64(2), int64(1), object(7)
memory usage: 34.1+ MB
```

Figure 5: Table showing the data type of all columns in the dataset

As seen in figure 5, the dataset contains heterogeneous feature types. A mix of categorical variables (object), Boolean (bool), and numerical (float64/int64) data necessitates that the data is pre-processed before fitting. Categorical data types must be converted with one-hot encoding before use as the model cannot interpret words for training.

Analysis of Predictive Power of Features

```
# Sample for better visuals
sample_vis = adverts.sample(2000, random_state=RANDOM_STATE)

# ---- Mileage vs Price ----
x_mileage = sample_vis["mileage"].dropna()
y_price = sample_vis.loc[x_mileage.index, "price"].dropna()

# Apply percentile bounds
x_low, x_high = x_mileage.quantile([lower_bound, upper_bound])
y_low, y_high = y_price.quantile([lower_bound, upper_bound])

plt.figure(figsize=(6,5))
plt.scatter(
```

```
    sample_vis.loc[x_mileage.index, "mileage"],
    sample_vis.loc[x_mileage.index, "price"],
    alpha=0.1
)
plt.xlim(x_low, x_high)
plt.ylim(y_low, y_high)
plt.xlabel("Mileage")
plt.ylabel("Price (£)")
plt.title("Mileage vs Price (sample, 1st–99th percentile)")
plt.show()
```

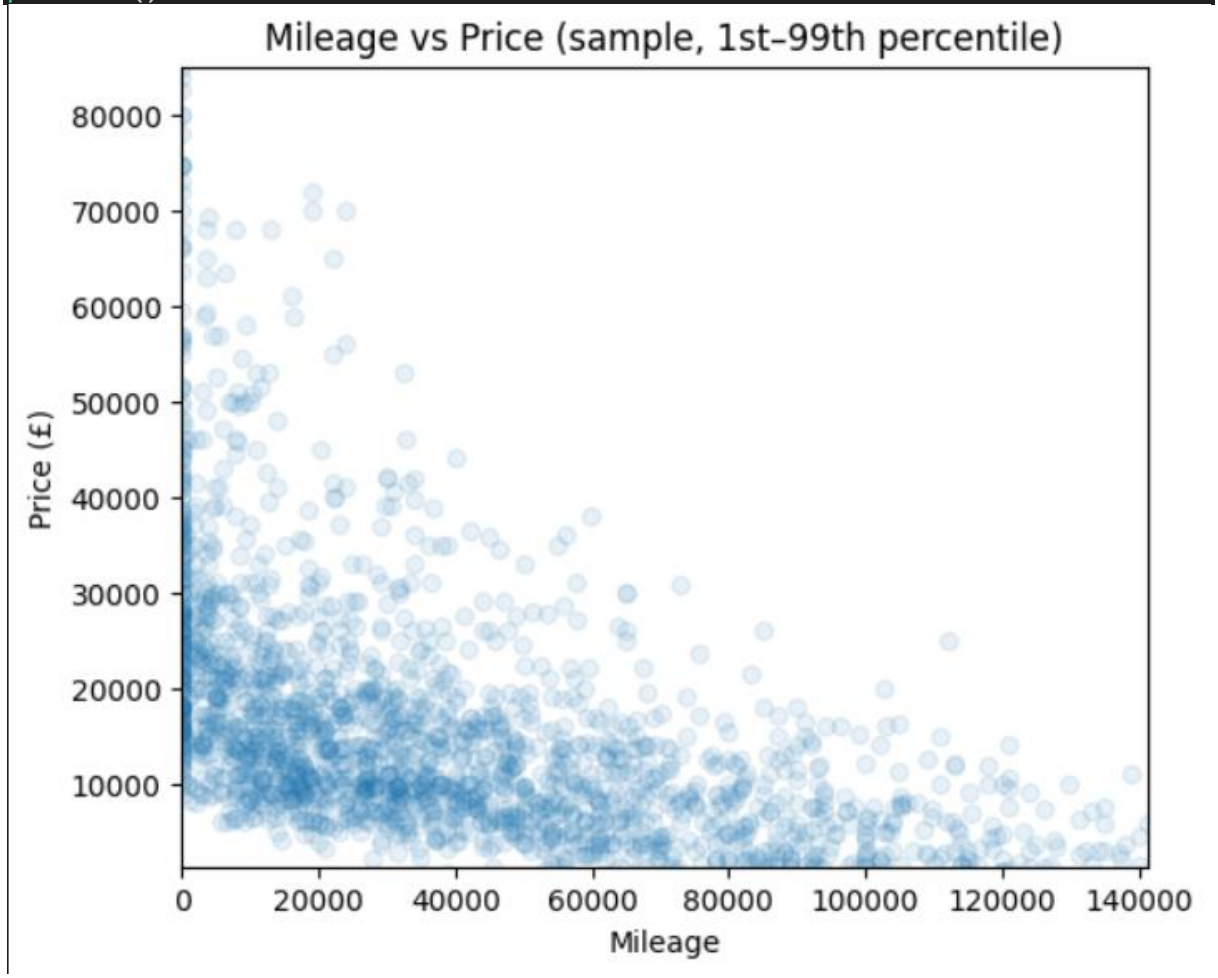

Figure 6: Mileage plotted against price (limited to 2000 entries between 1st-99th percentiles)

Mileage shows a strong negative relationship with selling price, where vehicles with higher mileage tend to be sold at lower prices. This aligns with domain expectations, as mileage is a proxy for vehicle wear and remaining lifespan.

Although it isn't exactly linear, the overall trend suggests that mileage can be a strong predictive feature and should have significant contribution to price estimation.

```
# ---- Year of registration vs Price ----
x_year = sample_vis["year_of_registration"].dropna()
y_price = sample_vis.loc[x_year.index, "price"].dropna()

# Apply percentile bounds
```

```
x_low, x_high = x_year.quantile([lower_bound, upper_bound])
y_low, y_high = y_price.quantile([lower_bound, upper_bound])

plt.figure(figsize=(6,5))
plt.scatter(
    sample_vis.loc[x_year.index, "year_of_registration"],
    sample_vis.loc[x_year.index, "price"],
    alpha=0.1
)
plt.xlim(x_low, x_high)
plt.ylim(y_low, y_high)
plt.xlabel("Year of registration")
plt.ylabel("Price (£)")
plt.title("Year of registration vs Price (sample, 1st–99th percentile)")
```

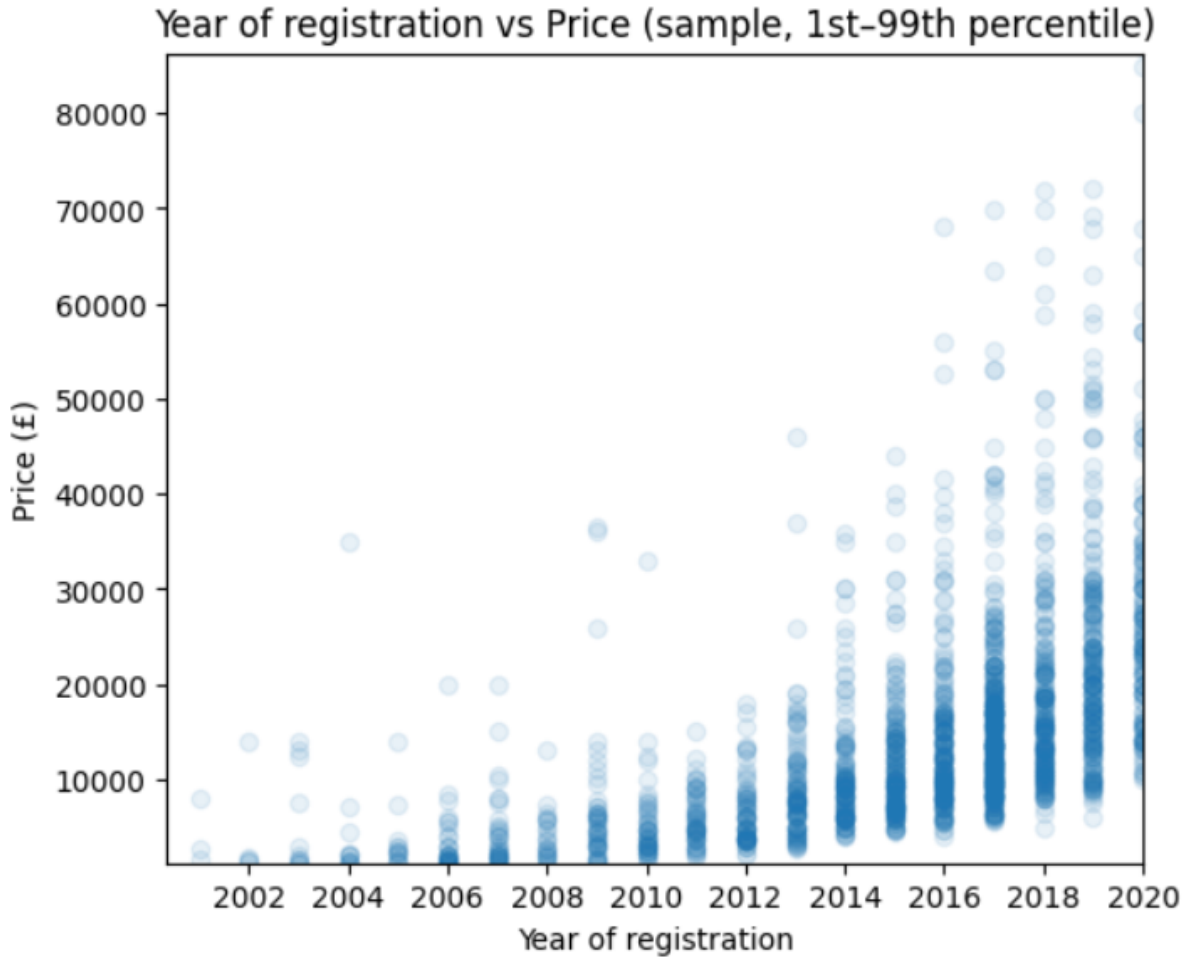

Figure 7: Year of registration plotted against price

The year of registration shows a positive relationship with selling price, where newer vehicles typically list for higher prices. This feature defines the vehicle age, which is very closely linked to depreciation and market value.

This clear trend suggests that year of registration is also an important predictor, although some dispersion remains due to factors such as the make, model, and the condition of the car.

```
# ---- Median price by key categorical predictors ----
```

```
for cat in ["standard_make", "fuel_type", "body_type", "vehicle_condition"]:
    if cat in adverts.columns:
        display(
            adverts
            .groupby(cat)["price"]
            .median()
            .sort_values(ascending=False)
            .head(15)
            .to_frame("median_price")
        )
```

standard_make	median_price
Pagani	2400000
Bugatti	1495000
Lamborghini	174990
Ferrari	145000
McLaren	137972
Rolls-Royce	130000
Radical	125950
BAC	106450
AC	94950
Maybach	82492
Lancia	74972
Aston Martin	74000
Datsun	69500
Corvette	66248
Bentley	65000

fuel_type	median_price
Diesel Hybrid	39990
Diesel Plug-in Hybrid	35991
Petrol Plug-in Hybrid	30995
Electric	27894
Petrol Hybrid	17814
Bi Fuel	14000
Diesel	13495
Petrol	11000
Natural Gas	3795

body_type	median_price
Camper	30000
Minibus	19500
Coupe	18000
SUV	17950
Pickup	17376
Saloon	16250
Combi Van	15125
Convertible	14900
Limousine	13995
Estate	13495
Panel Van	12995
Window Van	9995
MPV	9882
Hatchback	8995
Chassis Cab	8750

vehicle_condition	median_price
NEW	26760

USED	11890

Figure 8: Table showing median price based on different categorical features

From figure 8, we can see that the biggest factor for the selling price of a car is its make, as the median price is significantly higher with premium manufacturers.

When properly encoded, these categorical features can provide substantial pricing information. These price differences also coincide with what we expect in practice, and display a very intuitive relationship with selling price, especially newer cars being priced substantially higher than used cars.

Data Processing for Data Exploration and Visualisation

For exploratory purposes, the selling price was log-transformed (Figure 2) to reduce the strong right skew observed in the original distribution. This transformation improves the interpretability of visualisations by compressing the extreme values, and allowing patterns to be more clearly observed within the dataset.

Applying this transformation during exploration also informed the later decision to use a log-transformed target during model training

To better understand the relationship between the categorical features and price, the data was grouped by vehicle make and aggregated using the median price. Using the median price is a better indicator as extreme outliers can cause a skew if price was set as the mean.

This step made it easier to identify systematic price differences between manufacturers during exploratory analysis.

Given the large size of the dataset (about 400,000 listings), random subsampling was used to improve the visual clarity in scatter plots. Plotting the full dataset would obscure any meaningful pattern. Therefore, we can preserve the overall structure of the data while enabling better visual exploration

Data Processing for Machine Learning

Dealing with Missing Values, Outliers, and Noise

```
# dealing with missing values

# replace missing numeric values with median, and scale for Nearest Neighbour
and Linear Regression models.
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

# replace missing categorical cells with unknown
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="constant", fill_value="Unknown")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

# pass bool features as is because there are no missing values
preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
        ("bool", "passthrough", boolean_features),
    ],
    remainder="drop"
)
```

Missing values were handled using simple imputation strategies applied separately to numerical, categorical, and Boolean features.

Numerical variables were imputed using the median, which as previously mentioned, is more robust against outliers and skewed distributions, while categorical variables were set to unknown. This approach avoids introducing potentially incorrect assumptions about missing entries.

For Boolean feature type, there was no missing data and was passed through as is.

```
# Target variable
target = "price"

# Remove non-predictive identifier column
cleaned_ads = adverts.drop(columns=["public_reference"]).copy()

# --- Basic sanity / noise handling (rule-based) ---
# Keep only positive prices
before = len(cleaned_ads)
cleaned_ads = cleaned_ads[cleaned_ads[target].notna() & (cleaned_ads[target] >
0)]
```

```
print(f"Removed {before - len(cleaned_ads):,} rows with missing/non-positive
price.")
```

The public reference was cleaned from the dataset immediately as it is a unique identifier for the car for use by AutoTrader rather than something that influences the price. Keeping this identifier could have introduced noise without adding any predictive values and provide bogus results.

In addition to this, a check was done to ensure that any listings with no pricing or negative pricing was removed from the dataset. Although, this is unlikely to be the case as listings tend to have the price as a mandatory field with checks to ensure a correct number is entered.

```
y_pred = linreg.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
```

```
# log transforming price to deal with skewed pricing (new cars, luxury, etc.)
y_log = np.log1p(y)

y_pred_log = linreg_log.predict(X_test)

# transforming price to actual values
y_pred = np.expm1(y_pred_log)
y_true = np.expm1(y_test_log)
mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_true, y_pred)
```

```
MAE:  3,695    MAE (log model):  2,474
RMSE: 36,977   RMSE (log model): 36,942
R^2:  0.329    R^2 (log model):  0.330
```

Figure 9: Mean Absolute Error, Root Mean Square Error, and R^2 score of Linear Regression Model using raw values, and log-transformed values

Rather than explicitly removing high-priced vehicles, the effect of outliers in the price was mitigated using a logarithmic transformation. This approach reduces the influence of extreme values, while preserving all observations, which is preferable when high-value cases are valid albeit rare.

This transformation stabilises the variance and improved model robustness. As seen in figure 9, this transformation resulted in almost 30% reduction in the mean absolute error, indicating that it is a more viable method for fitting.

As linear regression was the first model to be trained, it is the only one that used raw values. KNN and Decision Tree utilise log-transformed values.

Feature Engineering, Data Transformations, Feature Selection

```
# replace missing categorical cells with unknown
```

```
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="constant", fill_value="Unknown")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])
```

Categorical features such as vehicle make, model, fuel type, and body type were transformed using one-hot encoding. Previously unseen categories are ignored to ensure that they do not cause errors, improving robustness.

One-hot encoding converts each category to a binary indicator variable, allowing the models to process it without imposing artificial ordinal relationships.

Numerical features were standardised using z-score scaling. This transformation places numerical variables on a comparable scale, which is particularly important for distance-based models such as KNN. Scaling also improves the numerical stability, and ensures that no single feature dominates due to its magnitude.

Model Building

Algorithm Selection, Model Instantiation and Configuration

Linear regression was selected as a strong baseline model due to its simplicity, interpretability, and efficiency on large datasets.

Training the model on log-transformed prices allowed it to better capture multiplicative relationships between features and selling price, while reducing the impact of extreme values. This model provides a useful reference point against which more flexible algorithms can be compared.

K-Nearest Neighbours (KNN) regression captures potential non-linear relationships in the data without making strong parametric assumptions. The model predicts prices based on the average of nearby observations in the feature space.

As KNN relies on distance calculations, feature scaling was used to ensure that numerical variables contributed proportionally. While flexible, KNN can be sensitive to noise and is computationally expensive on large datasets.

Decision Tree regression was the third and final model used. It can model non-linear relationships and interactions between features.

Trees can naturally handle complex decision boundaries and mixed feature types after encoding. However, it is prone to overfitting if not properly constrained using depth and sample size hyper-parameters.

Similarly to KNN, this regression model is computationally expensive, and was by far the slowest to train.

Grid Search, Model Ranking and Selection

```
param_grid_knn = {
    "model__n_neighbors": [5, 7, 9],
    "model__weights": ["uniform", "distance"] # 6 permutations
}
```

```
# A small-but-realistic grid to explore under/overfitting trade-offs
param_grid_tree = {
    "model__max_depth": [5, 10, 20],
    "model__min_samples_leaf": [1, 10, 20],
    "model__min_samples_split": [2, 10] # 18 permutations
}
```

For KNN and Decision Tree, a grid of plausible hyper-parameter values was defined to explore different levels of complexity and behaviour. Rather than relying on default settings, this approach enables systematic comparison of configurations, and minimises the risk of under or over-fitting.

While this can be effective, it is important to note that these parameters significantly increase the number of permutations within the model, which was minimised for speed, given the size of the dataset.

```
grid_knn = GridSearchCV(
    knn,
    param_grid_knn,
    scoring="neg_mean_absolute_error",
    cv=cv,
    n_jobs=processes
)
```

Grid search was combined with k-fold cross-validation to estimate model performance on unseen data. Mean absolute error was used as the optimisation metric due to its robustness to outliers and the direct interpretability in price units.

Cross-validation ensures that hyper-parameter selection is based on generalisation performance rather than a single train-validation split.

```
best_knn = grid_knn.best_estimator_
best_tree = grid_tree.best_estimator_
```

Models were ranked based on their cross-validated MAE scores, and the configuration with the best average performance was selected as the final model.

This approach enables a fair comparison across different algorithms and settings. The selected model was then evaluated on a held-out test set to obtain an unbiased estimate of predictive performance.

Model Evaluation and Analysis

Coarse-Grained Evaluation / Analysis

Linear Regression (raw): [MAE: 3695, RMSE: 36977, R^2: 0.329]
Linear Regression (log-transformed): [MAE: 2474, RMSE: 36942, R^2: 0.330]

Linear regression trained on raw prices achieves moderate performance, explaining approximately 1/3 of the variance in selling price. After applying a log transformation to the target, the mean absolute error is reduced substantially, indicating the improved average predictive accuracy across typical vehicles.

The similar R^2 values suggest that, while the transformation improves absolute error, overall variance remains dominated by high priced vehicles. This further highlights the effectiveness of log-transformation in improving robustness, without fundamentally altering the model capacity.

KNN: [Best parameters: (n_neighbours = 7, weights = distance), MAE: 3041, RMSE: 41572, R^2: 0.151]

The KNN model achieves reasonable average accuracy, but underperforms relative to linear regression. The lower R^2 indicates limited ability to explain overall price variance, while the high RMSE suggests increased sensitivity to outliers.

Despite tuning, KNN struggles with the high dimensionality introduced by one-hot encoded categorical features, that can dilute distance-based similarity in large feature spaces.

Decision Tree: [Best parameters: (max_depth = 20, min_samples_leaf = 10, min_samples_split = 2), MAE: 3198, RMSE: 40934, R^2: 0.177

The decision tree model captures non-linear relationships but does not outperform the linear baseline. Its MAE and R^2 indicate limited generalisation, similarly to KNN.

This suggests that even with depth and leaf size constraints, the tree may not effectively exploit the strong linear and categorical signals already captured by the linear regression model.

Overall, evaluation shows that the log-transformed linear regression provides the best balance of accuracy and explanatory power, justifying its selection as the preferred model for further analysis.

Feature Importance

The analysis shows that vehicle make and model contribute most strongly to the model's predictive performance. Disrupting these features may lead to the largest degradation in accuracy, indicating that the model relies heavily on them when estimating selling price.

This aligns with practical expectations, as make and model implicitly encode information about build quality, brand positioning, typical specifications, and market perception. Their dominance suggests that much of the price signal is captured through categorical identifiers rather than purely numerical statistics.

Mileage and year of registration provide complementary information related to vehicle usage and age, both of which are known drivers of depreciation. Although their individual importance is lower than that of make and model, they still contribute meaningfully to prediction accuracy.

This indicates that while categorical features capture the broad price differences, numerical features aid to refine predictions within specific vehicle categories.

Fine-Grained Evaluation

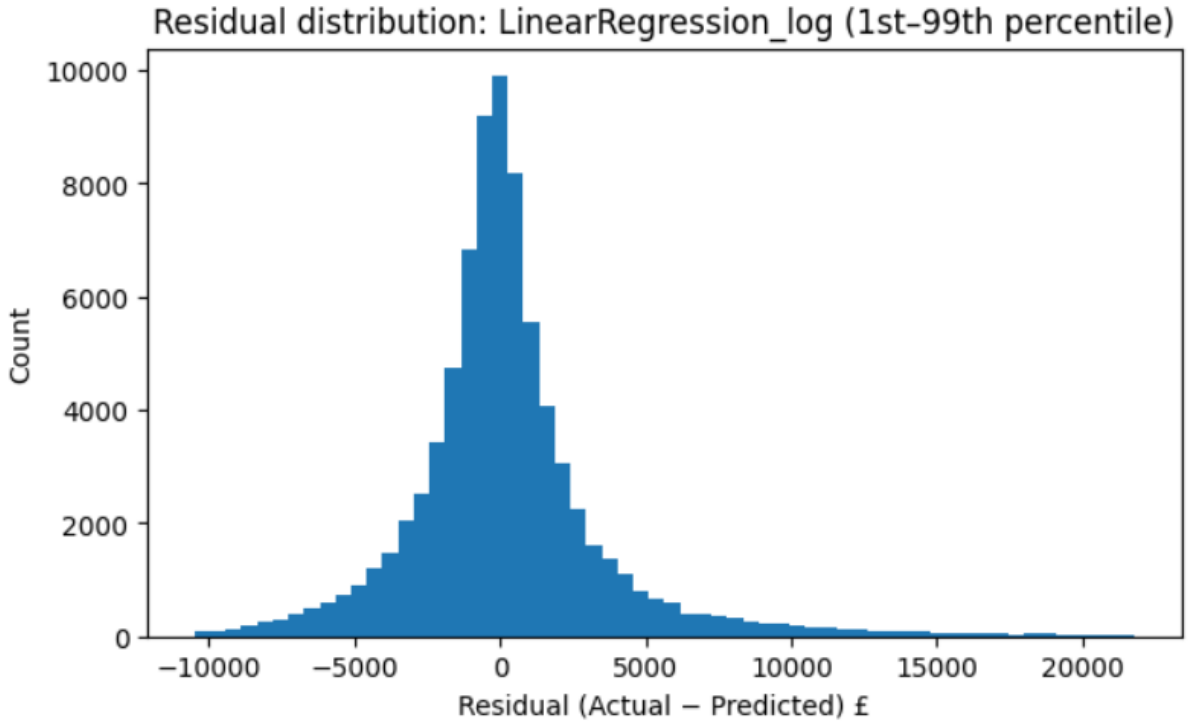

Figure 10: Residuals plotted against count

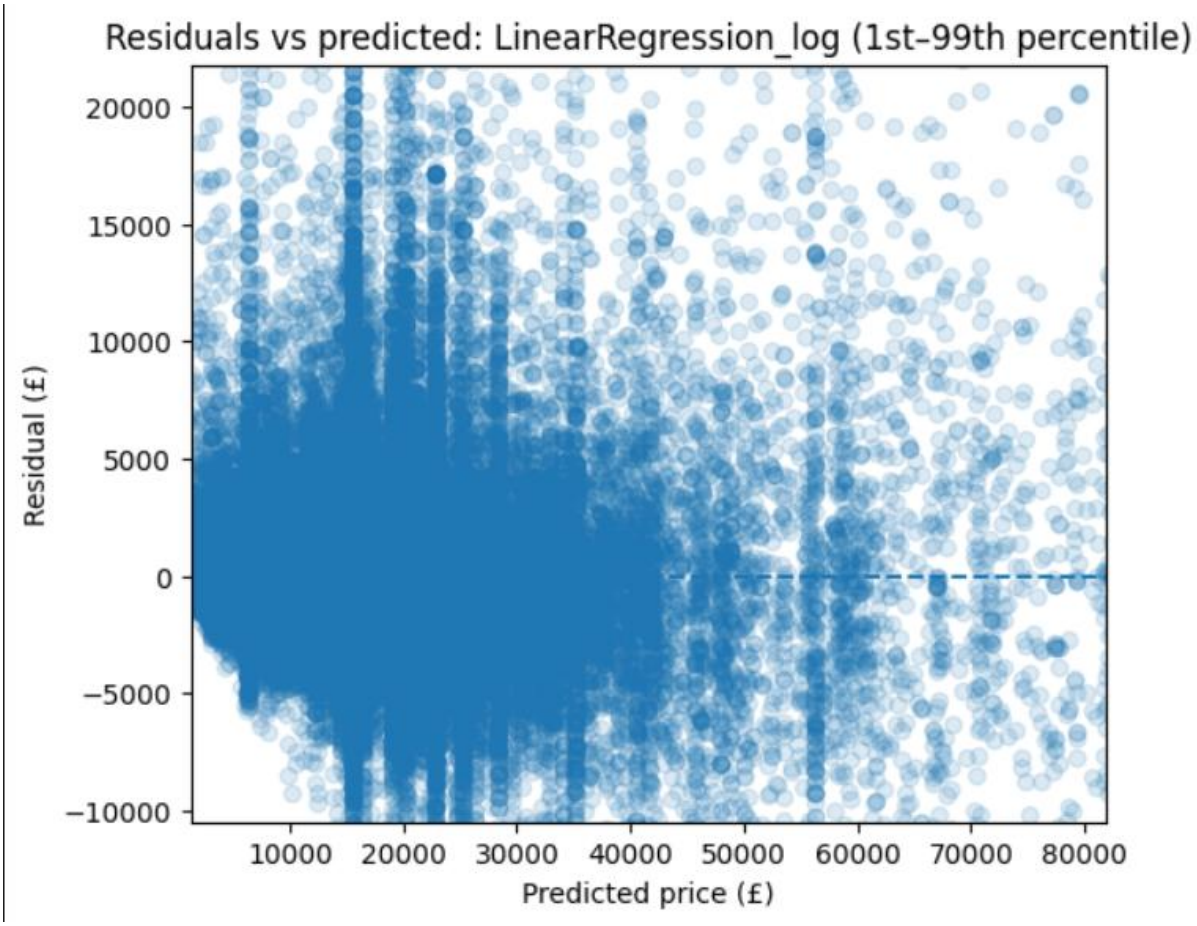

Figure 11: Predicted price plotted against residuals.

The residual distribution is centred close to zero and is approximately symmetric, indicating that the model does not exhibit strong systematic bias across the dataset.

Most prediction errors are relatively small, however, the presence of long tails on both sides of the distribution highlights that large errors still occur for a subset of observations.

These extreme residuals are likely associated with atypical or high-value cars, which are inherently more difficult to price accurately.

The scatter plot shows a large cluster, further expressing that typical vehicles are more easily priced, compared to the wider spread in residuals as the predicted price increases.

Together, these analyses show that the predictive reliability decreases for premium or unusual cars. This limitation is common in real-world pricing models and provides important context for interpreting aggregate performance metrics.