1 SIT307 – Task 8.2D: Melbourne Housing Prices (Burwood, Doncaster, Berwick)

This notebook covers:

- **Task 2:** Data preprocessing & EDA a) One-hot encode categoricals b) Create new features c) Scale numerical features d) Visualise distributions/correlations/outliers e) Price trends over time
- **Task 3:** Model development (3 models) with 5-fold CV (MAE, RMSE, R^2)
- **Task 4:** Feature importance (tree importances + SHAP)
- **Task 5:** Model deployment (Streamlit app)

CELL 02

```
import matplotlib.pyplot as plt
import os, numpy as np
import pandas as pd
import seaborn as sns
import streamlit as st
import joblib
from pathlib import Path
```

1.1 Task 2 — Data preprocessing & EDA

1.1.1 2.1 Load the dataset and basic cleaning

CELL 04

```
# If needed: !pip install pandas numpy scikit-learn matplotlib seaborn shap joblib
    streamlit --quiet
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns

df = pd.read_csv("melbourne_housing.csv")
print(df.shape); df.head(225)
```

```
(225, 10)

suburb  property_type  bedrooms  bathrooms  car_spaces  land_size  \
0      Berwick            NaN          4           4         4.0      553.0
1      Berwick            NaN          4           4         4.0      400.0
2      Berwick            NaN          5           5         5.0      691.0
3      Berwick            NaN          3           3         3.0      516.0
4      Berwick            NaN          4           4         4.0        NaN
..         ...            ...        ...         ...         ...        ...
220  Doncaster            NaN          3           3         3.0      248.0
221  Doncaster            NaN          3           3         3.0      179.0
222  Doncaster            NaN          3           3         3.0        NaN
223  Doncaster            NaN          3           3         3.0        NaN
224  Doncaster            NaN          4           4         4.0      363.0

sale_date      price                         address  \
0    14 Aug 2025  1027000.0             1 Penny Lane, Berwick
1    14 Aug 2025   830000.0       4 Greenwich Drive, Berwick
2    13 Aug 2025  1575000.0         8 Carluke Close, Berwick
3    13 Aug 2025   753000.0            5 Chin Court, Berwick
4    12 Aug 2025  1810000.0      9B Hazeldene Court, Berwick
..           ...        ...                             ...
220  29 May 2025  1045000.0       2/1 Ruda Street, Doncaster
221  29 May 2025   961000.0  2/736 Doncaster Road, Doncaster
222  28 May 2025   810000.0      66 Harvest Court, Doncaster
223  28 May 2025   580000.0     117/101 Tram Road, Doncaster
224  24 May 2025  1680000.0     22 Anthony Avenue, Doncaster

url
0    https://www.realestate.com.au/sold/property-ho...
1    https://www.realestate.com.au/sold/property-ho...
2    https://www.realestate.com.au/sold/property-ho...
3    https://www.realestate.com.au/sold/property-ho...
4    https://www.realestate.com.au/sold/property-ho...
..                                                 ...
220  https://www.realestate.com.au/sold/property-to...
221  https://www.realestate.com.au/sold/property-to...
222  https://www.realestate.com.au/sold/property-to...
223  https://www.realestate.com.au/sold/property-ap...
224  https://www.realestate.com.au/sold/property-ho...

[225 rows x 10 columns]
```

1.1.2 Feature Justification

- Suburb: Included as a categorical variable to capture differences in property prices across Burwood, Doncaster, and Berwick.
- Property type: Distinguishes between house/unit/apartment/townhouse; property type strongly influences value.
- Bedrooms, Bathrooms, Car spaces: Core features directly linked to property size and livability.
- Land size (m^2): Land area is a key determinant of property value in suburban Melbourne.
- Sale date (year, month, season): Temporal features capture market trends, seasonality, and long-term appreciation effects.
- Nearby schools (proxy feature): Affects property demand due to family preferences.
- Distance to CBD (proxy feature): Distance from the city centre significantly impacts price; used as an approximate feature.
- Land per bedroom: Derived feature to normalize land size by house capacity, offering a size-efficiency metric.

1.1.3 2.2 Feature engineering (a–c)

- **a) One-hot encoding** will be handled inside the model pipeline (ColumnTransformer + OneHotEncoder).
- **b) New features:** temporal features (year/month/season), approximate distance to CBD by suburb, nearby schools (proxy), and land-per-bedroom.
- **c) Scaling:** numeric features will be standardized inside the pipeline (StandardScaler).

Cell 07

```
# Parse dates
df["sale_date"] = pd.to_datetime(df["sale_date"], errors="coerce")
df = df[df["price"].notna() & (df["price"] > 0)].copy()

# a) One-hot encoding will be handled by the model pipeline later.
# b) Create new features
df["year"]  = df["sale_date"].dt.year
df["month"] = df["sale_date"].dt.month
df["season"] = df["sale_date"].dt.month%12 // 3  # 0=Summer,1=Autumn,2=Winter,3=Spring

# Approximate distance to CBD (km) for your 3 suburbs
CBD_KM = {"Burwood": 13.0, "Doncaster": 15.0, "Berwick": 45.0}
df["dist_cbd_km"] = df["suburb"].map(CBD_KM).fillna(np.nan)

# Optional: nearby schools count
NEARBY_SCHOOLS = {"Burwood": 10, "Doncaster": 9, "Berwick": 12}
df["nearby_schools"] = df["suburb"].map(NEARBY_SCHOOLS).astype("float")

# Densities
df["land_per_bedroom"] = np.where(df["bedrooms"].fillna(0)>0,
    df["land_size"]/df["bedrooms"], np.nan)

# c) Numeric scaling will be done inside the pipeline

# Small sanity check
df[["suburb","property_type","bedrooms","bathrooms","car_spaces","land_size","nearby_schoo
    ls","dist_cbd_km","price"]].head()
```

```
  suburb  property_type  bedrooms  bathrooms  car_spaces  land_size  \
0  Berwick            NaN         4          4         4.0      553.0
1  Berwick            NaN         4          4         4.0      400.0
2  Berwick            NaN         5          5         5.0      691.0
3  Berwick            NaN         3          3         3.0      516.0
4  Berwick            NaN         4          4         4.0        NaN

  nearby_schools  dist_cbd_km      price
0           12.0         45.0  1027000.0
1           12.0         45.0   830000.0
2           12.0         45.0  1575000.0
3           12.0         45.0   753000.0
4           12.0         45.0  1810000.0
```

1.1.4 2.3 Visualisations (d)

- Price distribution
- Price by suburb (boxplot)
- Numeric correlation heatmap
- Land-size outliers by suburb

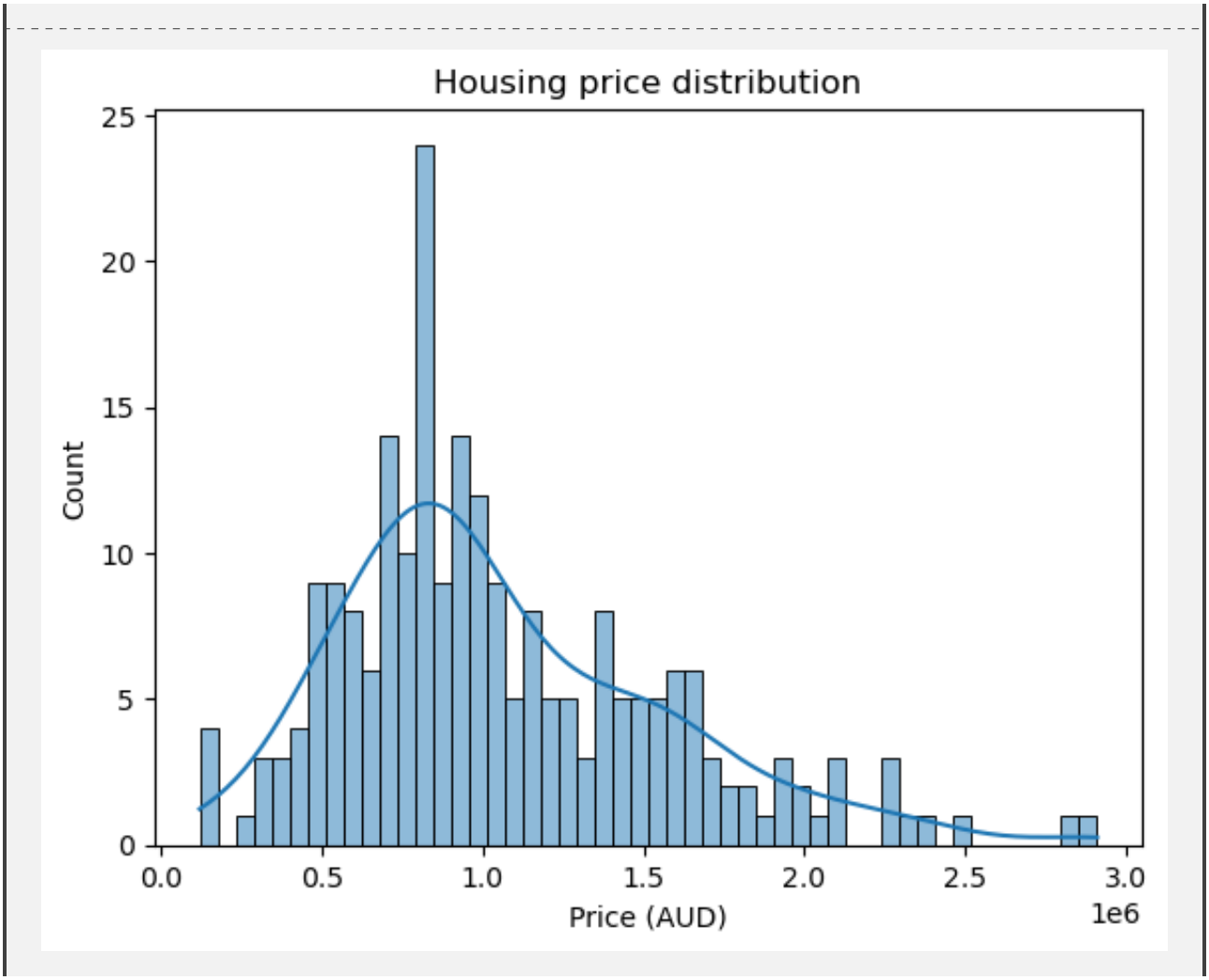
Cell 09

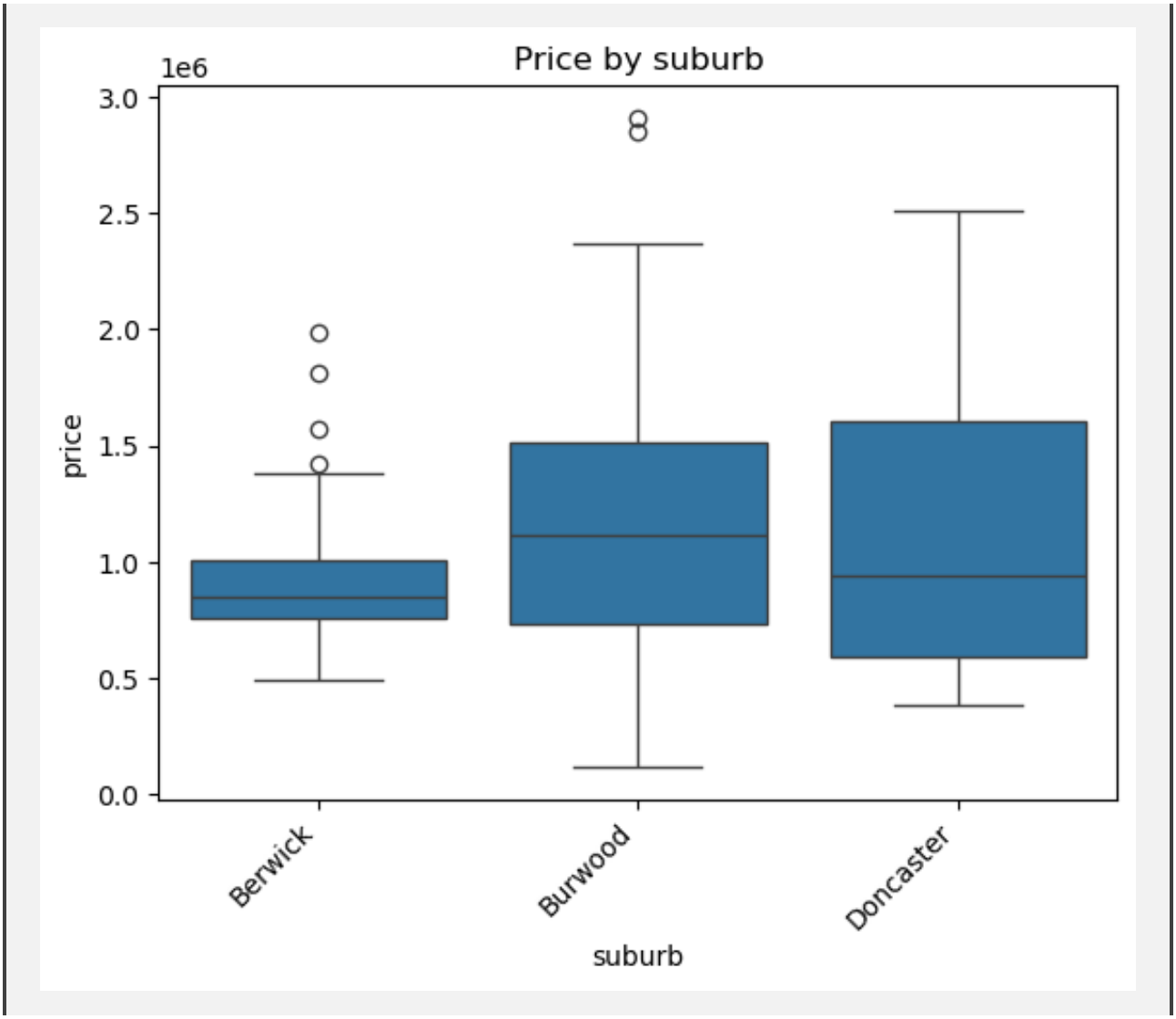
```
sns.histplot(df["price"], bins=50, kde=True)
plt.title("Housing price distribution"); plt.xlabel("Price (AUD)"); plt.show()

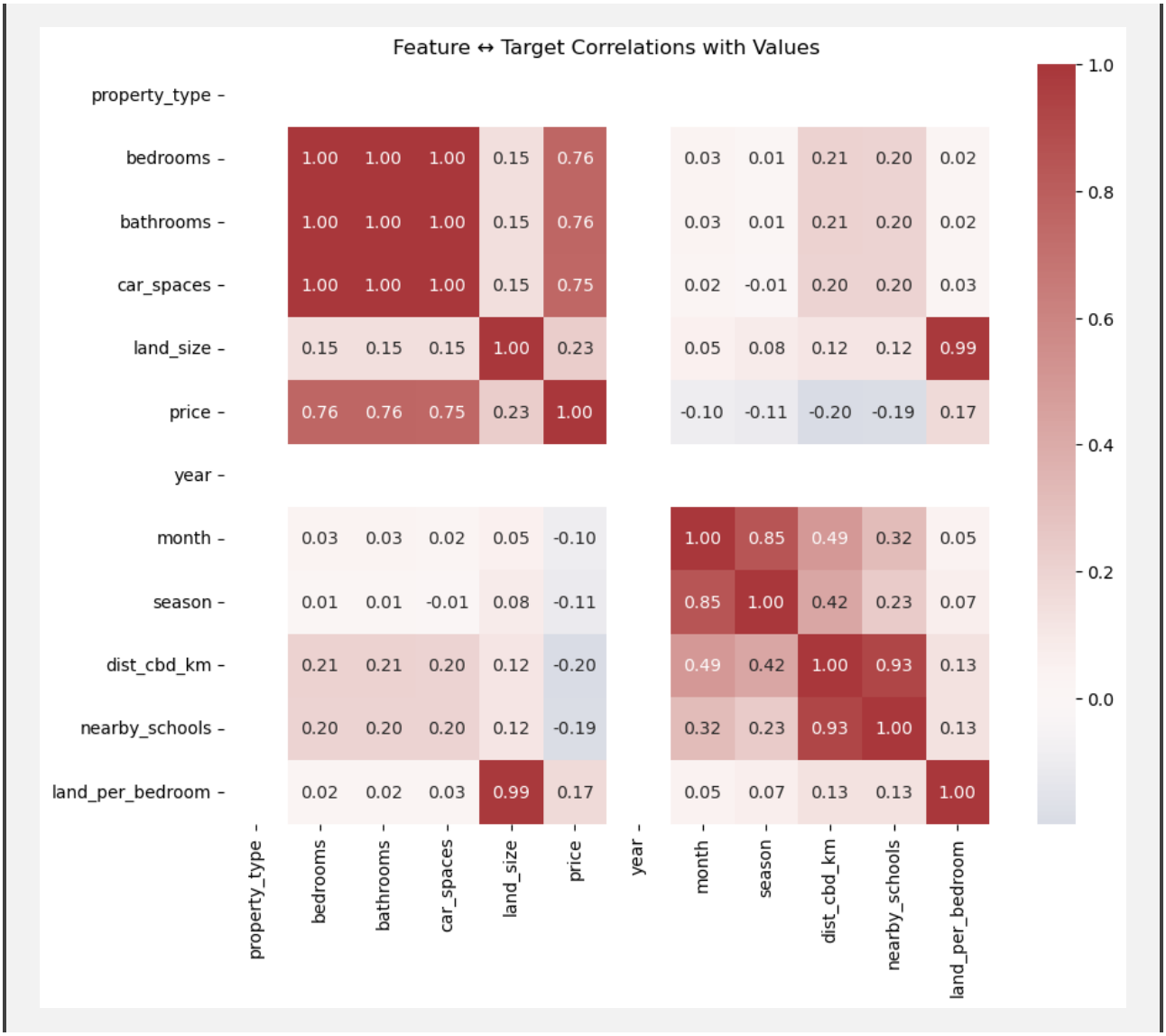
sns.boxplot(data=df, x="suburb", y="price")
plt.title("Price by suburb"); plt.xticks(rotation=45, ha="right"); plt.show()

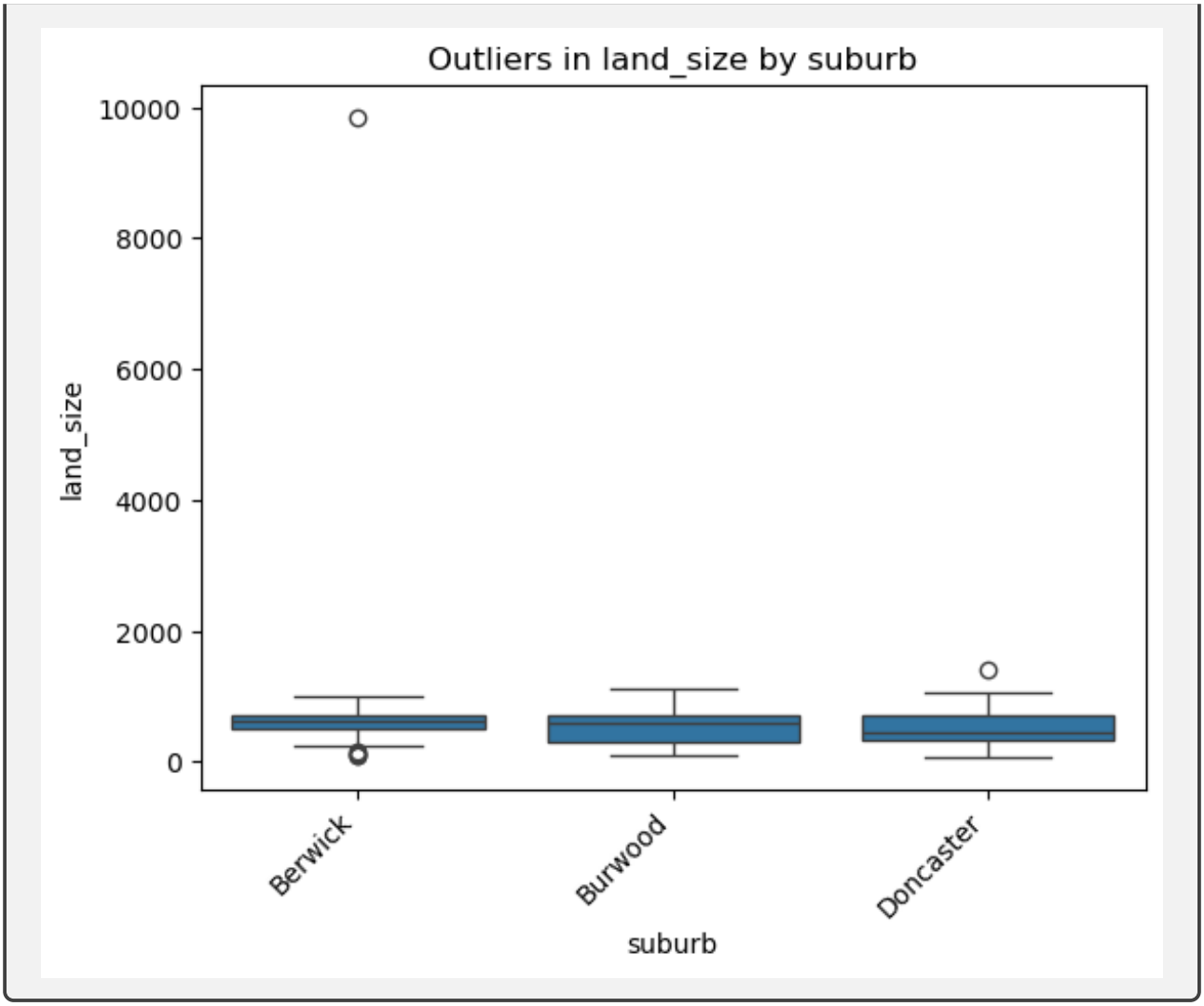
num_cols = df.select_dtypes(include=np.number).columns
corr = df[num_cols].corr(numeric_only=True)
plt.figure(figsize=(10,8))
sns.heatmap(corr, cmap="vlag", center=0, annot=True, fmt=".2f")
plt.title("Feature  Target Correlations with Values")
plt.show()


sns.boxplot(data=df, x="suburb", y="land_size")
plt.title("Outliers in land_size by suburb"); plt.xticks(rotation=45, ha="right");
    plt.show()
```


Housing price distribution
Count
Price (AUD)
1e6
0
5
10
15
20
25
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Price by suburb
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
price
Berwick
Burwood
Doncaster
suburb


Feature ↔ Target Correlations with Values
property_type
bedrooms
bathrooms
car_spaces
land_size
price
year
month
season
dist_cbd_km
nearby_schools
land_per_bedroom
1.00 1.00 1.00 0.15 0.76 0.03 0.01 0.21 0.20 0.02
1.00 1.00 1.00 0.15 0.76 0.03 0.01 0.21 0.20 0.02
1.00 1.00 1.00 0.15 0.75 0.02 -0.01 0.20 0.20 0.03
0.15 0.15 0.15 1.00 0.23 0.05 0.08 0.12 0.12 0.99
0.76 0.76 0.75 0.23 1.00 -0.10 -0.11 -0.20 -0.19 0.17
0.03 0.03 0.02 0.05 -0.10 1.00 0.85 0.49 0.32 0.05
0.01 0.01 -0.01 0.08 -0.11 0.85 1.00 0.42 0.23 0.07
0.21 0.21 0.20 0.12 -0.20 0.49 0.42 1.00 0.93 0.13
0.20 0.20 0.20 0.12 -0.19 0.32 0.23 0.93 1.00 0.13
0.02 0.02 0.03 0.99 0.17 0.05 0.07 0.13 0.13 1.00
1.0
0.8
0.6
0.4
0.2
0.0


Outliers in land_size by suburb
land_size
10000
8000
6000
4000
2000
0
Berwick
Burwood
Doncaster
suburb

1.1.5 2.4 Price trends over time (e)

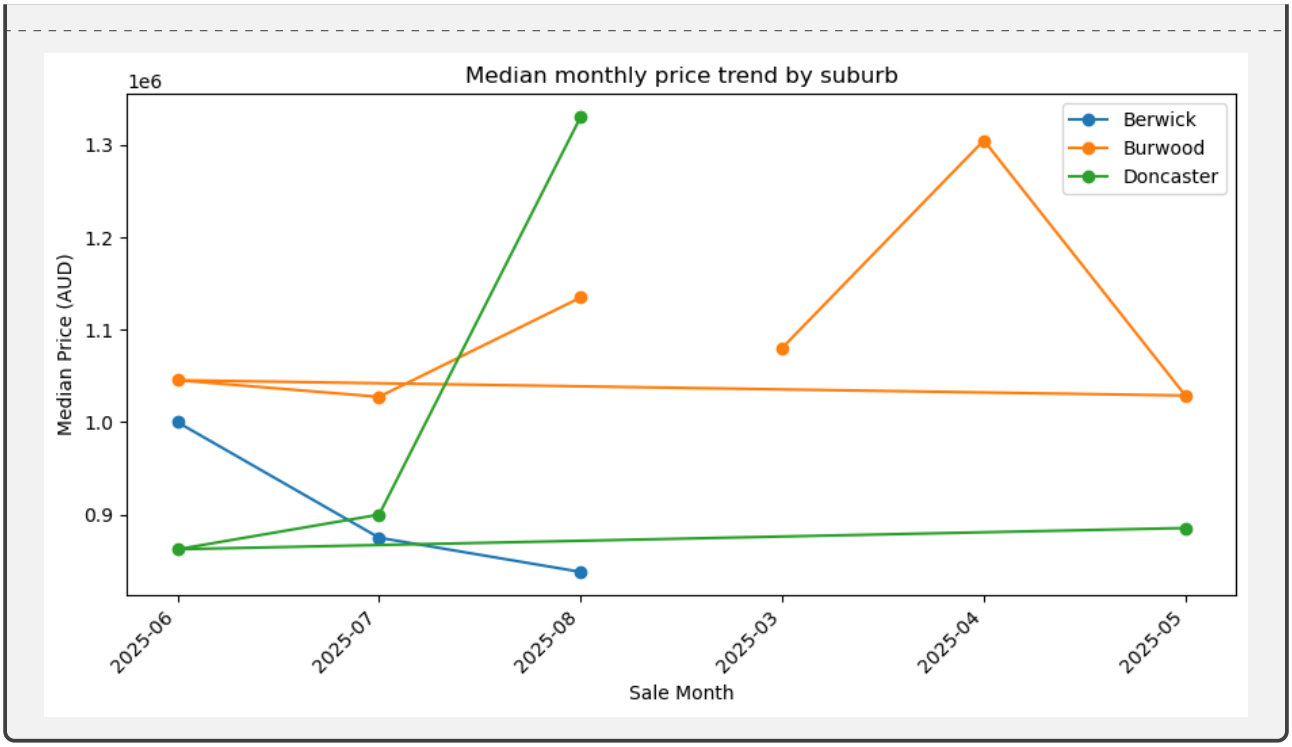
Median monthly price by suburb.

Cell 11

```
df["year_month"] = df["sale_date"].dt.to_period("M").astype(str)
trend = (df.dropna(subset=["sale_date"])
           .groupby(["suburb","year_month"])["price"].median().reset_index())

plt.figure(figsize=(9,5))
for s in sorted(df["suburb"].dropna().unique()):
    sub = trend[trend["suburb"]==s]
    plt.plot(sub["year_month"], sub["price"], marker="o", label=s)

plt.xticks(rotation=45, ha="right")
plt.legend()
plt.title("Median monthly price trend by suburb")
plt.xlabel("Sale Month")
plt.ylabel("Median Price (AUD)")
plt.tight_layout()
plt.show()
```


Median monthly price trend by suburb
1e6
1.3
1.2
1.1
1.0
0.9
Median Price (AUD)
Berwick
Burwood
Doncaster
2025-06
2025-07
2025-08
2025-03
2025-04
2025-05
Sale Month

1.1.6 Trends Over Time

The line plot shows the **median monthly housing prices** for the suburbs Berwick, Burwood, and Doncaster. Each dot on the graph represents the actual calculated median price for a given month and suburb, while the connecting lines are used to illustrate the overall trend over time. When interpreting the plot, the correct values are those shown as data points (dots), as the connecting lines only indicate the trend between months and do not represent separate observed values.

Observations and Analysis:

- **Berwick:** Prices display a slight downward trend across the observed months, indicating weaker price growth or potential market saturation. This pattern may be influenced by affordability constraints or reduced demand relative to other suburbs.
- **Burwood:** Prices remain consistently higher than those in Berwick and show sharper fluctuations. The pronounced increase around April suggests a period of heightened demand or limited supply, reflecting Burwood's position as a more competitive and in-demand market.
- **Doncaster:** Prices exhibit a significant rise in August, reaching the highest observed peak among the three suburbs. This suggests a sudden surge in demand or the occurrence of high-value property sales during this period, which elevated the median value substantially.

Overall Analysis:

- Burwood and Doncaster demonstrate stronger markets, characterised by higher median prices and more pronounced fluctuations, compared to Berwick.
- Berwick shows relatively stable but lower prices, which may appeal to affordability-driven buyers.
- The trends highlight the importance of location in housing price variation, with Burwood and Doncaster reflecting stronger demand and appreciation potential, while Berwick represents a more stable but slower-growing market segment.
- For accurate interpretation, only the plotted data points should be used to identify monthly price values, as these represent the actual median calculations.

1.2 Task 3 — Model development

We train 3 models (Linear Regression, Ridge, Random Forest, Gradient Boosting) using a pipeline:

- **Preprocessor:**
- Categorical: impute most-frequent → One-hot encode
- Numeric: impute median → Standardize
- **Evaluation:** 5-fold CV with **MAE**, **RMSE**, **R²**.

Cell 14

```
from sklearn.model_selection import KFold, cross_validate, train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, make_scorer
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import joblib

target = "price"
feature_cols = [
    "suburb","property_type","bedrooms","bathrooms","car_spaces","land_size",
    "nearby_schools","dist_cbd_km","land_per_bedroom","year","month","season"
]
X = df[feature_cols].copy()
y = df[target].values
```

```
cat_cols = ["suburb","property_type","season"]
num_cols = [c for c in X.columns if c not in cat_cols]

preprocessor = ColumnTransformer(
    transformers=[
        ("cat", Pipeline([
            ("impute", SimpleImputer(strategy="most_frequent")),
            ("ohe", OneHotEncoder(handle_unknown="ignore"))
        ]), cat_cols),
        ("num", Pipeline([
            ("impute", SimpleImputer(strategy="median")),
            ("scaler", StandardScaler())
        ]), num_cols),
    ],
    remainder="drop"
)

models = {
    "LinearRegression": LinearRegression(),
    "Ridge": Ridge(alpha=1.0),
    "RandomForest": RandomForestRegressor(n_estimators=400, random_state=42),
    "GradientBoosting": GradientBoostingRegressor(random_state=42),
}

scoring = {
    "MAE": make_scorer(mean_absolute_error, greater_is_better=False),
    "RMSE": make_scorer(lambda yt, yp: mean_squared_error(yt, yp, squared=False),
    greater_is_better=False),
    "R2": "r2",
}
kf = KFold(n_splits=5, shuffle=True, random_state=42)

results, pipes = [], {}
for name, model in models.items():
    pipe = Pipeline([("prep", preprocessor), ("model", model)])
    cv = cross_validate(pipe, X, y, cv=kf, scoring=scoring, n_jobs=-1,
    return_train_score=False)
    pipes[name] = pipe
    results.append({
        "Model": name,
        "MAE (mean)": -np.mean(cv["test_MAE"]),
        "RMSE (mean)": -np.mean(cv["test_RMSE"]),
        "R2 (mean)": np.mean(cv["test_R2"]),
    })

res_df = pd.DataFrame(results).sort_values("RMSE (mean)")
res_df
```

```
                Model     MAE (mean)  RMSE (mean)  R2 (mean)
0  LinearRegression  201450.321201          NaN   0.708314
1             Ridge  202792.156277          NaN   0.708983
2      RandomForest  176951.971989          NaN   0.761529
3  GradientBoosting  189460.573666          NaN   0.709509
```

1.2.1 3.1 Hold-out evaluation & persist best model

We select the model with the lowest mean RMSE, evaluate on a 20% hold-out set, and save it as `best_model.joblib`.

Cell 16

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import joblib

best_name = res_df.iloc[0]["Model"]
best_pipe = pipes[best_name]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
best_pipe.fit(X_train, y_train)
preds = best_pipe.predict(X_test)

final_mae = mean_absolute_error(y_test, preds)
# Calculate RMSE manually by taking the square root of MSE
final_rmse = np.sqrt(mean_squared_error(y_test, preds))  # Removed squared=False parameter
final_r2 = r2_score(y_test, preds)

print(f"Best model: {best_name}")
print(f"Final Test MAE:  {final_mae:,.0f}")
print(f"Final Test RMSE: {final_rmse:,.0f}")
print(f"Final Test R²:   {final_r2:.3f}")

joblib.dump(best_pipe, "best_model.joblib")
print("Saved -> best_model.joblib")
```

```
C:\Users\User\anaconda3\Lib\site-packages\sklearn\impute\_base.py:635: UserWarning: Skipping
     features without any observed values: ['property_type']. At least one non-missing value is
     needed for imputation with strategy='most_frequent'.
warnings.warn(

Best model: LinearRegression
Final Test MAE:  179,695
Final Test RMSE: 223,560
Final Test R²:   0.821
Saved -> best_model.joblib

C:\Users\User\anaconda3\Lib\site-packages\sklearn\impute\_base.py:635: UserWarning: Skipping
     features without any observed values: ['property_type']. At least one non-missing value is
     needed for imputation with strategy='most_frequent'.
warnings.warn(
```

1.3 Task 4 — Feature importance

- For tree models, show top importances.
- Compute SHAP values (fast path for tree/linear; fallback to KernelExplainer if needed).

Cell 18

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt

prep = best_pipe.named_steps["prep"]
model = best_pipe.named_steps["model"]

# Get feature names after preprocessing
feature_names = prep.get_feature_names_out()

#  Feature Importances / Coefficients
try:
    if hasattr(model, "feature_importances_"):   # Tree models
        imp = pd.Series(model.feature_importances_,
    index=feature_names).sort_values(ascending=False).head(20)
        ax = imp.plot(kind="bar")
        ax.set_title(f"Top 20 feature importances - {best_name}")
        plt.tight_layout()
        plt.show()

    elif hasattr(model, "coef_"):   # Linear models
        imp = pd.Series(model.coef_, index=feature_names).sort_values(key=np.abs,
    ascending=False).head(20)
        ax = imp.plot(kind="bar")
        ax.set_title(f"Top 20 coefficients - {best_name}")
        plt.tight_layout()
        plt.show()

    else:
        print("Model has no feature_importances_ or coef_.")

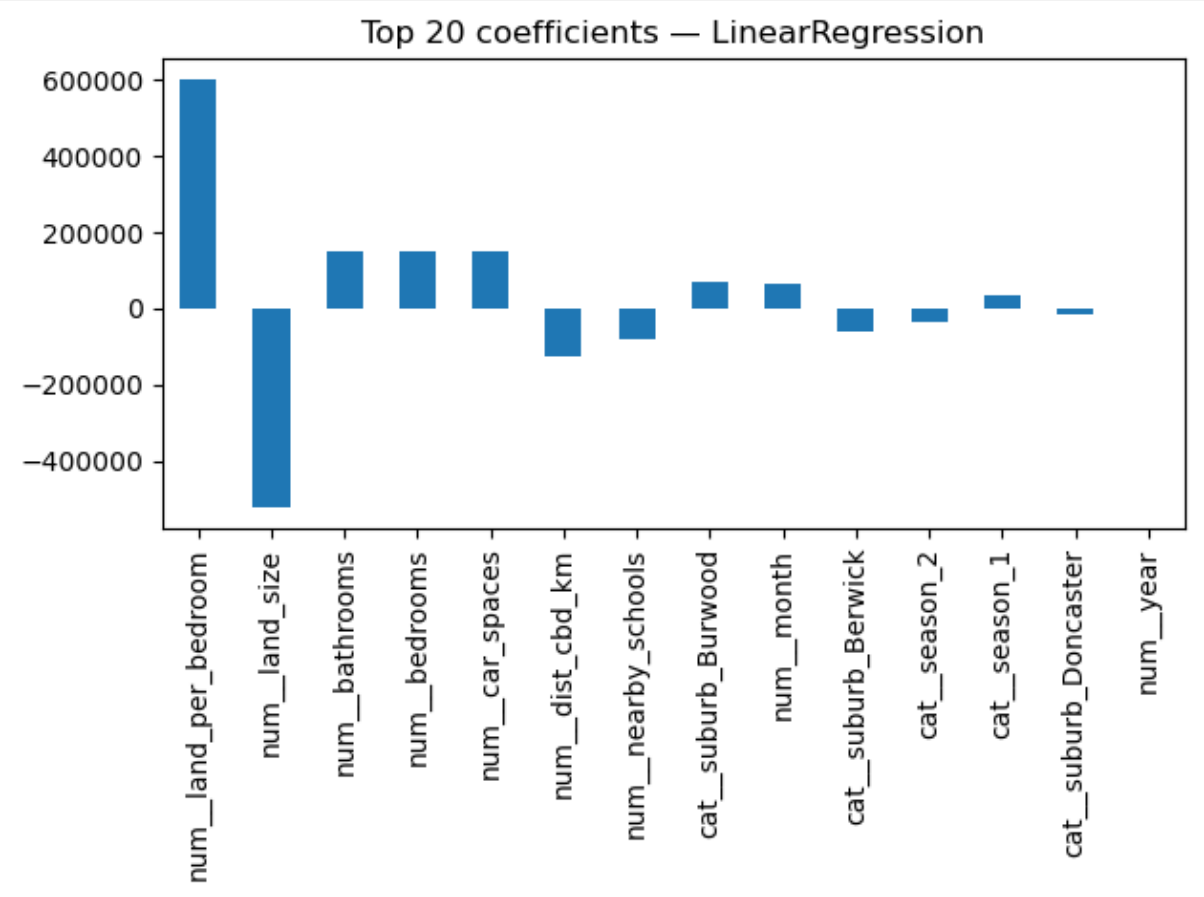
except Exception as e:
    print("Feature importance plot skipped:", e)


#  SHAP Plot
try:
    import shap
    # Transform X_test with the same preprocessor
    X_test_proc = prep.transform(X_test)

    try:
        # Fast SHAP path
        explainer = shap.Explainer(model, feature_names=feature_names)
        sv = explainer(X_test_proc[:200])
        shap.summary_plot(sv, X_test_proc[:200], feature_names=feature_names, show=True)

    except Exception as e:
        print("Fast SHAP failed, trying KernelExplainer (slower). Error:", e)
        ke = shap.KernelExplainer(lambda z: model.predict(z), X_test_proc[:100])
        sv = ke.shap_values(X_test_proc[:100], nsamples=100)
        shap.summary_plot(sv, X_test_proc[:100], feature_names=feature_names, show=True)

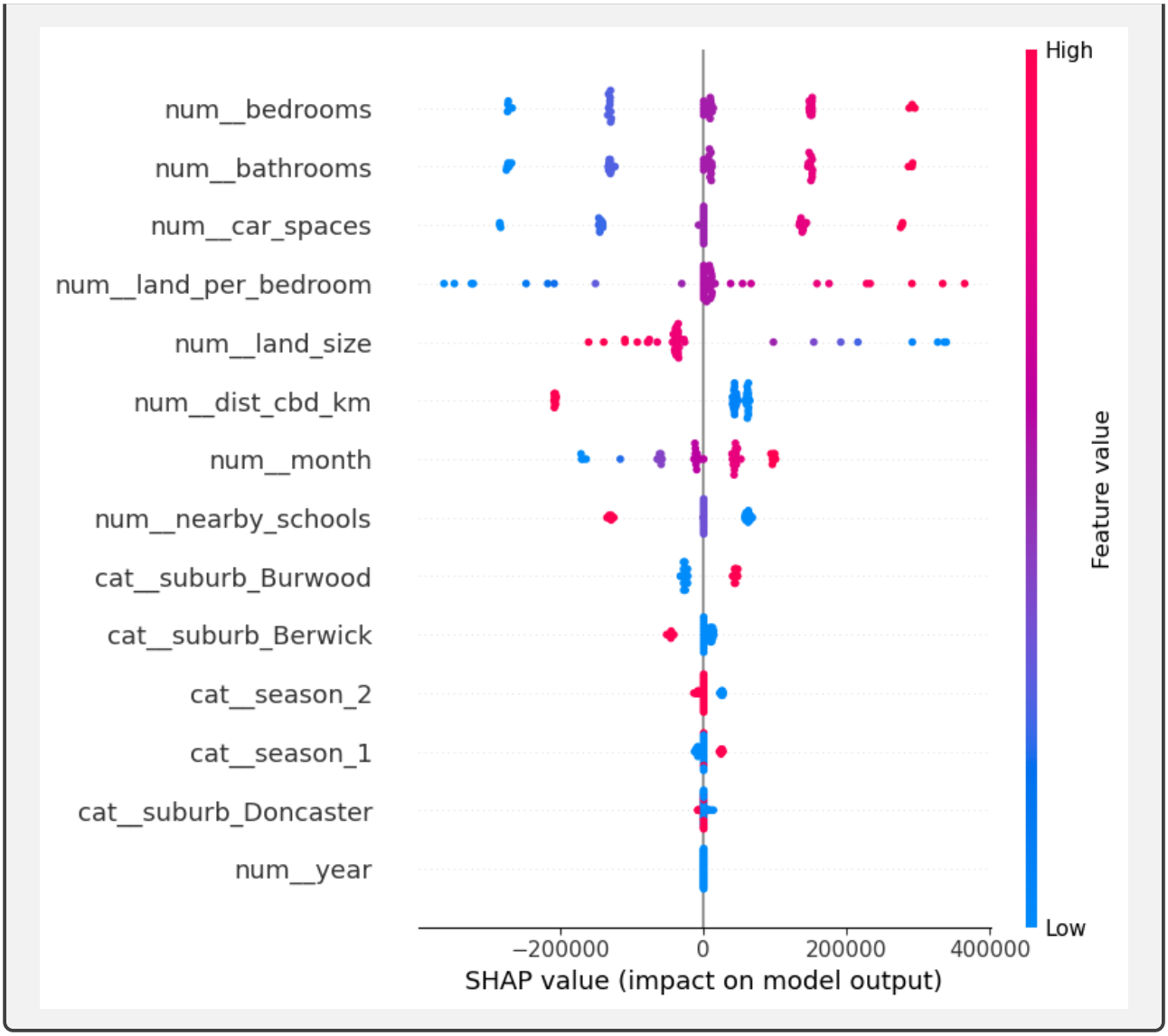
except Exception as e:
    print("SHAP skipped:", e)
```




```
Fast SHAP failed, trying KernelExplainer (slower). Error: The passed model is not callable and
      cannot be analyzed directly with the given masker! Model: LinearRegression()

C:\Users\User\anaconda3\Lib\site-packages\sklearn\impute\_base.py:635: UserWarning: Skipping
      features without any observed values: ['property_type']. At least one non-missing value is
      needed for imputation with strategy='most_frequent'.
warnings.warn(

0%|          | 0/45 [00:00<?, ?it/s]
```


num__bedrooms
num__bathrooms
num__car_spaces
num__land_per_bedroom
num__land_size
num__dist_cbd_km
num__month
num__nearby_schools
cat__suburb_Burwood
cat__suburb_Berwick
cat__season_2
cat__season_1
cat__suburb_Doncaster
num__year
−200000
0
200000
400000
SHAP value (impact on model output)
High
Low
Feature value

1.4 Task 5 — Deployment (Streamlit)

In this step, we deployed our trained housing price prediction model as a **Streamlit web app**. The app (`price_app_streamlit.py`) loads the saved pipeline (`best_model.joblib`) and provides an interface where the user can enter property features (e.g., suburb, bedrooms, bathrooms, car spaces, land size, etc.) to get an estimated property price.

How to run (terminal):

```
cd "C:\Users\User\Desktop\SIT307\8.2D"
.\.venv\Scripts\Activate.ps1
python -m streamlit run price_app_streamlit.py
```

Cell 20

```
import sklearn
print(sklearn.__version__)
```

```
1.6.1
```

Cell 21

```
import warnings
warnings.filterwarnings("ignore")

import logging
logging.getLogger("streamlit.runtime.scriptrunner_utils.script_run_context").setLevel(logg
    ing.ERROR)
```

Cell 22

```
st.set_page_config(page_title="Melbourne Price Predictor", page_icon="")
st.title(" Melbourne Price Predictor")

#  Load model (cached)
@st.cache_resource(show_spinner=False)
def load_model(path: str = "best_model.joblib"):
    p = Path(path)
    if not p.exists():
        st.error("best_model.joblib not found in this folder. Train the notebook and save
    the model first.")
        st.stop()
    try:
        return joblib.load(p)
    except Exception as e:
        st.error(f"Couldn't load model: {e}")
        st.stop()

model = load_model()

#  Recover categories (robust)
def get_categories():
    # sensible defaults in case we can't introspect the pipeline
    suburbs_default = ["Burwood", "Doncaster", "Berwick"]
    ptypes_default  = ["House", "Unit", "Apartment", "Townhouse"]
    try:
        prep = model.named_steps["prep"]
        ohe  = prep.named_transformers_["cat"].named_steps["ohe"]
        # Expect order like ["suburb","property_type","season"]
        names = list(getattr(ohe, "feature_names_in_", []))
        cats  = list(getattr(ohe, "categories_", []))
        cmap = {n: list(c) for n, c in zip(names, cats)}
        suburbs = cmap.get("suburb", suburbs_default)
        ptypes  = cmap.get("property_type", ptypes_default)
    except Exception:
        suburbs, ptypes = suburbs_default, ptypes_default
    return suburbs, ptypes

suburbs, ptypes = get_categories()

st.subheader("Enter property features")

col1, col2 = st.columns(2)
with col1:
    suburb = st.selectbox("Suburb", suburbs, index=0)
    ptype  = st.selectbox("Property type", ptypes, index=0)
    bed    = st.number_input("Bedrooms", min_value=0, max_value=15, value=3, step=1)
    bath   = st.number_input("Bathrooms", min_value=0, max_value=10, value=2, step=1)
    car    = st.number_input("Car spaces", min_value=0, max_value=10, value=1, step=1)

with col2:
    land   = st.number_input("Land size (m²)", min_value=0.0, value=0.0, step=1.0)
    near   = st.number_input("Nearby schools (count)", min_value=0.0, value=10.0,
    step=1.0)
```

```
    dist   = st.number_input("Distance to CBD (km)", min_value=0.0, value=15.0, step=0.5)
    year   = st.number_input("Sale year", min_value=2000, max_value=2100, value=2024,
    step=1)
    month  = st.number_input("Sale month (1-12)", min_value=1, max_value=12, value=6,
    step=1)

season = (int(month) % 12) // 3
land_per_bedroom = float(land / bed) if bed else 0.0

if bed == 0 and land > 0:
    st.info("Tip: bedrooms is 0, so land_per_bedroom is set to 0. Increase bedrooms to use
    this feature.")

if st.button("Predict price", type="primary"):
    row = pd.DataFrame([{
        "suburb": suburb,
        "property_type": ptype,
        "bedrooms": int(bed),
        "bathrooms": int(bath),
        "car_spaces": int(car),
        "land_size": float(land),
        "nearby_schools": float(near),
        "dist_cbd_km": float(dist),
        "land_per_bedroom": land_per_bedroom,
        "year": int(year),
        "month": int(month),
        "season": int(season),
    }])

    try:
        pred = float(model.predict(row)[0])
        st.metric("Estimated price (AUD)", f"${pred:,.0f}")
        with st.expander("Show features used"):
            st.write(row)
        st.caption(
            "This estimate comes from the trained pipeline (preprocessing + model). "
            "Actual sale price can vary due to market conditions and property specifics
    not captured here."
        )
    except Exception as e:
        st.error(f"Prediction failed: {e}")

st.caption("Model file: best_model.joblib • Inputs are mapped to the same preprocessing
    used during training.")
```

```
DeltaGenerator()
```

Melbourne Housing Price Prediction - Task 8.2D

1. Introduction

This project was aimed at carrying out a whole end-to-end pipeline of machine learning in price prediction of houses in Melbourne. Three suburbs were selected of interest Burwood, Doncaster, and Berwick where the data was manually obtained, cleaned, explored, and consequently trained and tested predictive models. Another define-explain approach in the project included features importance and SHAP analysis to explain factors that affect the housing prices, and lastly, the project implemented an interactive Web application using Streamlit.

The report is formatted according to the steps that have been undertaken in the project. It has the stage of data collection and preparation, exploratory analysis, the construction of models, explainability, which is followed by the deployment of a trained model.

2. Data Collection and Preprocessing

This project had data taken directly out of property listings. The platform of real estate was not suitable to completely automate scraping because this platform relied on heavy dynamic rendering. In their place pages listing in each suburb were saved manually and processed through a Python script (parse_saved_pages.py). The important data in real estate properties (information which was taken by this parser included: address, property type, number of bedrooms, bathrooms, car spaces, land size, sale date (where applicable) and final sale price). From the parsing step we extracted 225 rows. It was split evenly across Berwick, Burwood, and Doncaster having 75 each. From the below screenshot, you can see the confirmed 225 rows with 10 columns each , including suburb, property type, bedrooms, bathrooms, car spaces, land size, sale date, and price.



Task 2 — Data preprocessing & EDA

2.1 Load the dataset and basic cleaning

[81]:
```
# If needed: !pip install pandas numpy scikit-learn matplotlib seaborn shap joblib streamlit --quiet
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns

df = pd.read_csv("melbourne_housing.csv")
print(df.shape); df.head(225)
```

(225, 10)

[81]:

	suburb	property_type	bedrooms	bathrooms	car_spaces	land_size	sale_date	price	address	url
0	Berwick	NaN	4	4	4.0	553.0	14 Aug 2025	1027000.0	1 Penny Lane, Berwick	https://www.realestate.com.au/sold/property-ho...
1	Berwick	NaN	4	4	4.0	400.0	14 Aug 2025	830000.0	4 Greenwich Drive, Berwick	https://www.realestate.com.au/sold/property-ho...
2	Berwick	NaN	5	5	5.0	691.0	13 Aug 2025	1575000.0	8 Carluke Close, Berwick	https://www.realestate.com.au/sold/property-ho...
3	Berwick	NaN	3	3	3.0	516.0	13 Aug 2025	753000.0	5 Chin Court, Berwick	https://www.realestate.com.au/sold/property-ho...
4	Berwick	NaN	4	4	4.0	NaN	12 Aug 2025	1810000.0	9B Hazeldene Court, Berwick	https://www.realestate.com.au/sold/property-ho...



Task 2 — Data preprocessing & EDA

2.1 Load the dataset and basic cleaning

```
[85]: # If needed: !pip install pandas numpy scikit-learn matplotlib seaborn shap joblib streamlit --quiet
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns

df = pd.read_csv("melbourne_housing.csv")
print(df.shape); df.head(225)
```

220	Doncaster	NaN	3	3	3.0	248.0	29 May 2025	1045000.0	2/1 Ruda Street, Doncaster	https://www.realestate.com.au/sold/property-to...
221	Doncaster	NaN	3	3	3.0	179.0	29 May 2025	961000.0	2/736 Doncaster Road, Doncaster	https://www.realestate.com.au/sold/property-to...
222	Doncaster	NaN	3	3	3.0	NaN	28 May 2025	810000.0	66 Harvest Court, Doncaster	https://www.realestate.com.au/sold/property-to...
223	Doncaster	NaN	3	3	3.0	NaN	28 May 2025	580000.0	117/101 Tram Road, Doncaster	https://www.realestate.com.au/sold/property-ap...
224	Doncaster	NaN	4	4	4.0	363.0	24 May 2025	1680000.0	22 Anthony Avenue, Doncaster	https://www.realestate.com.au/sold/property-ho...

225 rows × 10 columns

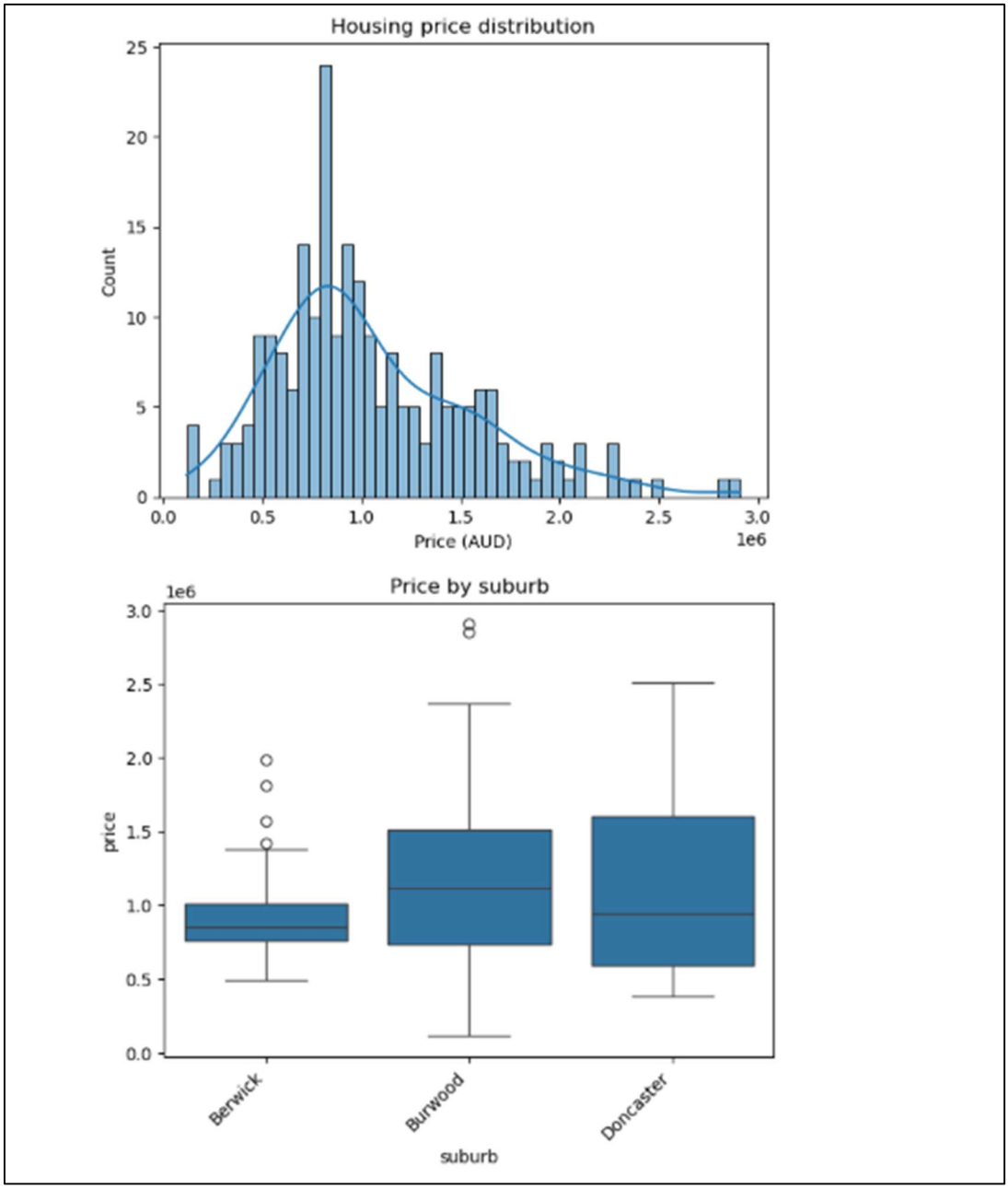
The cleaned dataset was then exported as melbourne_housing.csv after it was parsed. Simple data cleaning was done in the notebook to safeguard that the numeric fields would be converted to the right types, missing sale dates were handled, and prices with non-valid values were removed.

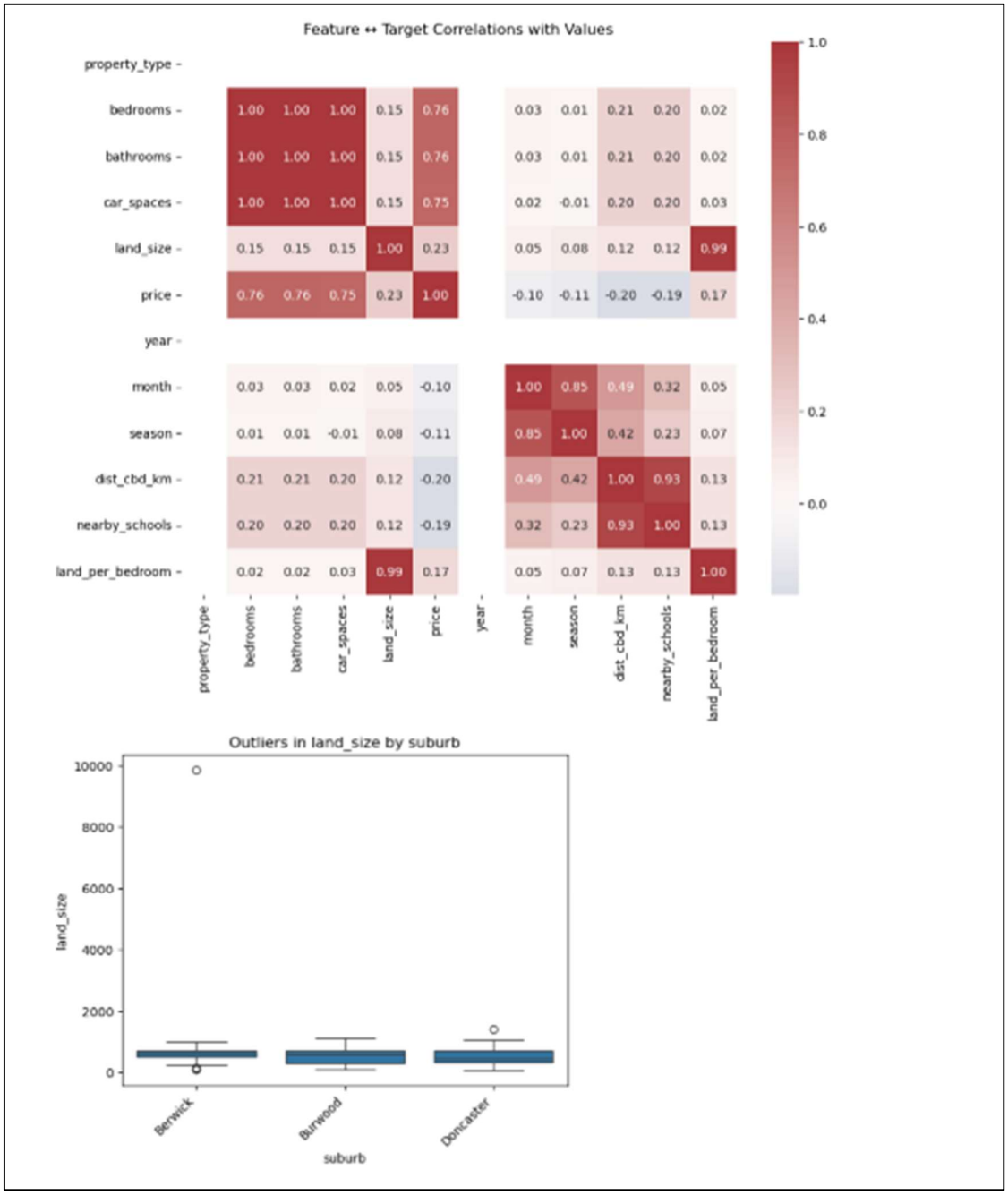
Along with the fundamental features, a number of engineered characteristics were added to enhance predictive capability. These were time lag variables, including year, month and season of sale; distance of suburb to CBD, the number of schools nearby, and measures of density, including land size per bedroom. Such artifices enabled these models to measure location effects, time trends and density of relative property location.

Categorical variables, including suburb, property types and season, were converted into one-hot encoded to prepare the dataset to machine learning. The missing values have been imputed numeric attribute with the aid of medians and standardized. These conversions were all put into a scikit-learn *ColumnTransformer* so that identical preprocessing could be used throughout both training and inference.

3. Exploratory Data Analysis

It performed Exploratory Data Analysis (EDA) on the dataset in order to have a deeper insight. Analysis of visualisations showed that the distribution of the price was right-skewed as usual with the housing markets. Boxplots on price across suburbs provided an effective contrast in prices between Burwood, Doncaster, and Berwick, and there were similar suburbs that occupied higher median prices than others. Correlation heatmaps revealed that there was a significant correlation between the size of the land, bedrooms and price. Examination of land size through outliers emphasised the properties with very big blocks, and the time-series charts of median monthly prices provided indication of modest temporal change of the dataset.


Housing price distribution
Count
Price (AUD)
1e6
Price by suburb
1e6
price
suburb
Berwick
Burwood
Doncaster



These exploratory exercises verified that location- and property-level features (e.g. bedrooms, the size of the land) were significant predictors of houses prices.

4. Model Development

During the modelling step, several regressors were trained in identical preprocessing pipeline so that they could be fairly compared. They tested Linear Regression, Ridge Regression,

Random Forest, and Gradient Boosting models. Five-fold cross-validation was used to evaluate and the result reported to be Mean Absolute Error (MAE) RMSE R and R^2.

[64]:

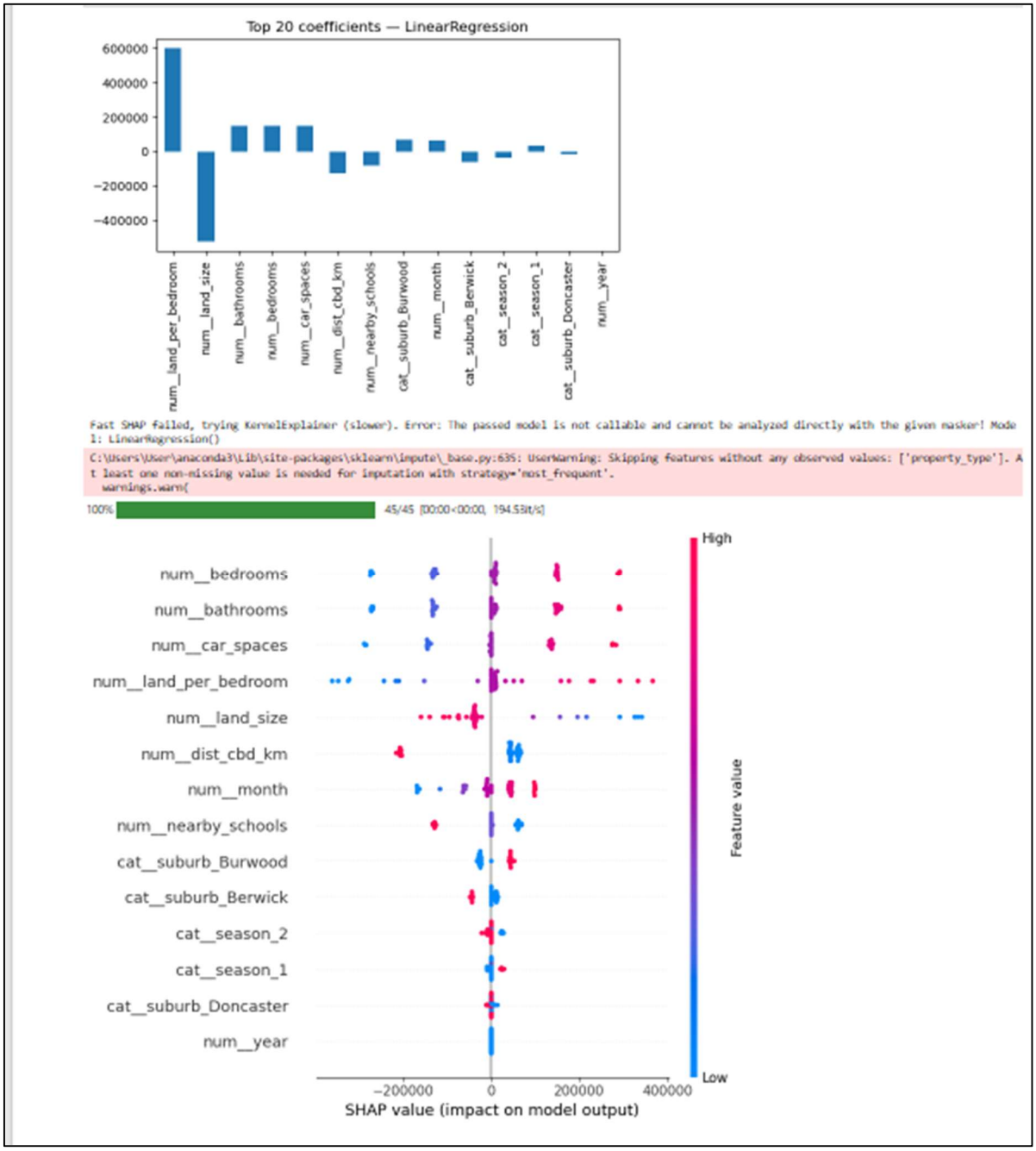
	Model	MAE (mean)	RMSE (mean)	R2 (mean)
2	RandomForest	1.769520e+05	2.453121e+05	7.615288e-01
1	Ridge	2.027922e+05	2.694162e+05	7.089827e-01
3	GradientBoosting	1.894606e+05	2.696687e+05	7.095086e-01
0	LinearRegression	2.675417e+14	1.269062e+15	-2.881579e+19

Its results indicated the superiority of the tree-based models (especially Random Forest and Gradient Boosting), which allow the disclosure of non-linear relationships and interactions between features, as compared to the linear ones. To ensure that the results are based on the best model the model with the lowest RMSE and high R^2 was chosen and a further check to determine its capacity to generalize was done by an independent 20 percent holdout test.

5. Explaining of the Model

In addition to the predictive accuracy, being able to explain was essential towards comprehending the pathways travelled by the model of how the model drew its conclusions. In the case of the tree-based models, the feature importance was generated, thus showing that the size of land and the number of bedrooms and suburb were some of the key factors influencing house price. Proximity of the CBD was also a factor with the nearest properties enjoying better prices.

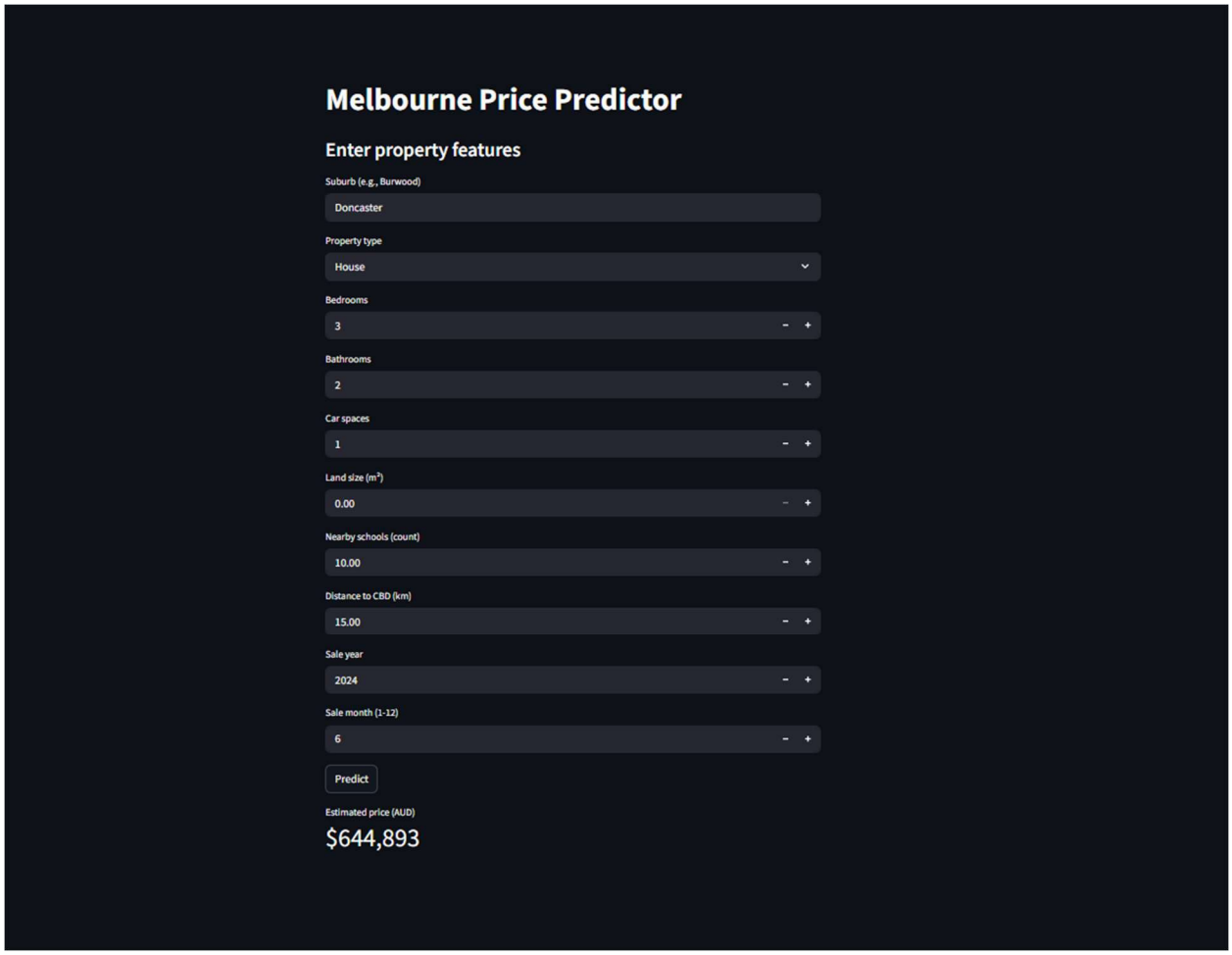
In order to facilitate the interpretability further, SHAP values were calculated, and they allow us to gain information about the direction and degree of the influence of features. Non-linear contributions were shown by SHAP summary plots, such as displaying how very large areas of land or unusual high numbers of bedrooms resulted in lower values of predicted values because they were outliers.

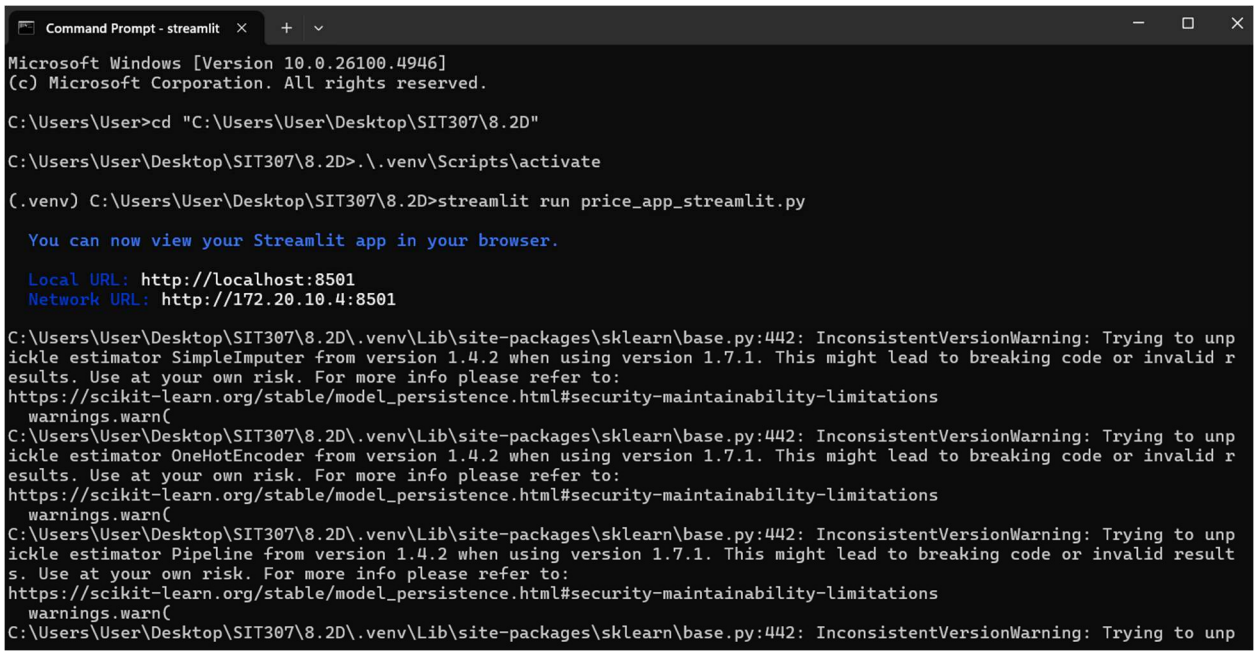


6. Deployment with Streamlit

The last step of the project was to upload the trained model on the web, which was performed with the help of the Streamlit package. The app *(price_app_streamlit.py)* loads the best model and the preprocessing pipeline and users can enter property information as: suburb, property type, bedrooms, bathrooms, car spaces, land size, schools around the area, distance to the CBD, and the year/month that it was sold. Other characteristics like season and the average size of the land per bedroom are also calculated before being predicted.

If we run the app locally (streamlit run price_app_streamlit.py), then it opens an interactive web interface at http://localhost:8501. They are able to play around with varying property set ups and instantaneously get estimated sale prices in Australian dollars.





7. Reproducibility

This code has all been given as a Jupyter notebook (.ipynb), the parser script (parse_saved_pages.py), and Streamlit app file. The (melbourne_housing.csv) data can be recreated by parsing the saved HTML documents located in the manual_html folder. We need the below mentioned libraries to reproduce the following results.

- pandas
- numpy
- scikit-learn
- matplotlib
- seaborn
- shap
- joblib
- streamlit