National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Aditya Pandey
Student ID: x23275286

School of Computing
National College of Ireland

Supervisor: Eric Gyamfi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Aditya Pandey
Student ID:	x23275286
Programme:	Data Analytics
Year:	2025
Module:	MSc Research Project
Supervisor:	Eric Gyamfi
Submission Due Date:	11/08/2025
Project Title:	Configuration Manual
Word Count:	371
Page Count:	9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Aditya Pandey
Date:	14th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	✓
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	✓
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	✓

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aditya Pandey
x23275286

1 Required Software and Tools

This project uses a Python-based data science stack. The core software environment should be set up as follows:

- **Python:** Version 3.8 or higher. It is recommended to use a virtual environment tool such as `conda` or `venv` to manage dependencies.
- **JupyterLab:** The analysis and model training were performed in a JupyterLab environment. Screenshots of the notebook are assumed to be a primary component of the manual's visual guide.

2 Project and Directory Structure

The codebase is organized in a clear, hierarchical structure. To ensure that file paths and dependencies resolve correctly, the following directory structure must be recreated:

```
airbnb-price-prediction-thesis/
├── data/
│   ├── listings.csv
│   └── reviews.csv
├── notebooks/
│   └── code.ipynb
├── models/
│   ├── multimodal_model_clean.pkl
│   ├── tabular_model_clean.pkl
│   ├── preprocessor_clean.pkl
│   └── streamlit_complete_model.json
├── src/
│   ├── streamlit_app.py
│   └── config.py
├── docs/
│   └── images/
│       └── [thesis charts \& plots]
├── requirements.txt
└── README.md
```

3 Python Library Dependencies

Python libraries can be installed using pip from a requirements.txt file or manually. subsection*Libraries Used

- pandas
- numpy
- scikit-learn (sklearn)
- scipy
- matplotlib
- seaborn
- transformers
- torch
- shap
- joblib
- pickle
- json

Figure 1: Necessary import modules

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, ExtraTreesRegressor, VotingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder, PowerTransformer, QuantileTransformer, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, r2_score
from scipy.stats import skew
import warnings
warnings.filterwarnings('ignore')
import pickle
import json
import os
from datetime import datetime
import re
import joblib

# Text processing imports
from transformers import DistilBertTokenizer, DistilBertModel
import torch
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
import shap
```

4 Execution and Replication Steps

The process involves data loading, feature engineering, model training (both tabular and multimodal), evaluation, and finally, generating visualizations and exporting model files.

4.1 Data Preprocessing and Feature Engineering

This section covers the initial steps of preparing the data for the models.

- **Load and Clean Data**: The code first loads the `listings.csv` and `reviews.csv` files into `pandas` DataFrames.

Figure 2: Data loading & preprocessing

```
# ========================================================================
# 4. DATA LOADING AND PREPROCESSING
# ========================================================================

print("Loading data...")

# Load data from organized structure
listings_path = os.path.join(data_dir, 'listings.csv')
reviews_path = os.path.join(data_dir, 'reviews.csv')

df = pd.read_csv(listings_path)
reviews_df = pd.read_csv(reviews_path)

print(f"Loaded {len(df)} listings and {len(reviews_df)} reviews")

# Aggregate review data by listing_id
review_aggregated = reviews_df.groupby('listing_id').agg({
    'comments': lambda x: ' '.join(x.dropna().astype(str)) if len(x.dropna()) > 0 else '',
    'id': 'count'
}).reset_index()
review_aggregated.columns = ['id', 'combined_reviews', 'review_count']

# Merge with listings data
df = df.merge(review_aggregated, on='id', how='left')
df['combined_reviews'] = df['combined_reviews'].fillna('')
df['review_count'] = df['review_count'].fillna(0)

print(f"After merging: {len(df)} listings with review data")
```

Figure 3: Setup file directory

```
# ========================================================================
# 2. SETUP PATHS AND DIRECTORIES
# ========================================================================

# Set up paths for organized project structure
project_root = os.path.dirname(os.getcwd())  # Go up one level from notebooks/
data_dir = os.path.join(project_root, "data")
models_dir = os.path.join(project_root, "models")

print(f"Project root: {project_root}")
print(f"Data directory: {data_dir}")
print(f"Models directory: {models_dir}")

# Ensure models directory exists
os.makedirs(models_dir, exist_ok=True)
os.makedirs('model_artifacts', exist_ok=True)
```

- **Feature Engineering**: The raw data is transformed into features suitable for the models. This includes processing the amenities and description text fields.

Figure 4: Feature engineering

```python
# Apply log transformation if skewed
y_skewness = skew(df['price_clean'].dropna())
if abs(y_skewness) > 1:
    df['price_clean'] = np.log1p(df['price_clean'])

# Create derived features
if 'accommodates' in df.columns:
    df['price_per_person'] = df['price_clean'] / df['accommodates'].replace(0, 1)

if 'bedrooms' in df.columns and 'beds' in df.columns:
    df['beds_per_bedroom'] = df['beds'] / df['bedrooms'].replace(0, 1)

if 'bathrooms_text' in df.columns:
    df['bathrooms_numeric'] = df['bathrooms_text'].str.extract(r'(\d+\.?\d*)').astype(float)

if 'neighbourhood_cleansed' in df.columns:
    neighbourhood_counts = df['neighbourhood_cleansed'].value_counts()
    df['neighbourhood_popularity'] = df['neighbourhood_cleansed'].map(neighbourhood_counts)

if 'host_is_superhost' in df.columns:
    df['is_superhost_numeric'] = (df['host_is_superhost'] == 't').astype(int)

if 'amenities' in df.columns:
    df['amenities_count'] = df['amenities'].str.count(',') + 1
    df['amenities_count'] = df['amenities_count'].fillna(0)

    # Key amenities
    key_amenities = ['wifi', 'kitchen', 'parking', 'pool']
    for amenity in key_amenities:
        df[f'has_{amenity}'] = df['amenities'].str.lower().str.contains(amenity, na=False).astype(int)
```

4.2 Model Training and Evaluation

This part of the code trains the various models and evaluates their performance.

Figure 5: Data splitting

```python
# ============================================================================
# 6. TRAIN-TEST SPLIT AND NEIGHBORHOOD ENCODING
# ============================================================================

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Add neighborhood-based features
if 'neighbourhood_cleansed' in X_train.columns:
    train_with_target = X_train.copy()
    train_with_target['target'] = y_train
    neighborhood_stats = train_with_target.groupby('neighbourhood_cleansed')['target'].agg(['mean', 'std']).reset_index()
    neighborhood_stats.columns = ['neighbourhood_cleansed', 'neighborhood_avg_price', 'neighborhood_price_std']

    X_train = X_train.merge(neighborhood_stats, on='neighbourhood_cleansed', how='left')
    X_test = X_test.merge(neighborhood_stats, on='neighbourhood_cleansed', how='left')

    overall_median = neighborhood_stats['neighborhood_avg_price'].median()
    overall_std = neighborhood_stats['neighborhood_price_std'].median()

    X_train['neighborhood_avg_price'] = X_train['neighborhood_avg_price'].fillna(overall_median)
    X_train['neighborhood_price_std'] = X_train['neighborhood_price_std'].fillna(overall_std)
    X_test['neighborhood_avg_price'] = X_test['neighborhood_avg_price'].fillna(overall_median)
    X_test['neighborhood_price_std'] = X_test['neighborhood_price_std'].fillna(overall_std)
```

Figure 6: Model training

```python
# ============================================================================
# 8. MODEL TRAINING
# ============================================================================

print("Training ensemble models...")

# Define models
models = {
    'ExtraTreesUltra': ExtraTreesRegressor(
        n_estimators=500, max_depth=25, min_samples_split=2,
        min_samples_leaf=1, max_features='sqrt', bootstrap=True,
        oob_score=True, random_state=42, n_jobs=-1
    ),
    'GradientBoostingUltra': GradientBoostingRegressor(
        n_estimators=500, learning_rate=0.05, max_depth=7,
        min_samples_split=10, min_samples_leaf=4, subsample=0.8,
        max_features='sqrt', random_state=42
    ),
    'RandomForestUltra': RandomForestRegressor(
        n_estimators=500, max_depth=30, min_samples_split=5,
        min_samples_leaf=2, max_features='sqrt', bootstrap=True,
        oob_score=True, random_state=42, n_jobs=-1
    )
}
# Train individual models
trained_models = {}
for name, model in models.items():
    pipeline = Pipeline([('preprocessor', preprocessor), ('regressor', model)])
    pipeline.fit(X_train, y_train)
    trained_models[name] = pipeline
    print(f"Trained {name}")

# Create ensemble
ensemble = VotingRegressor(estimators=[(name, model) for name, model in trained_models.items()], n_jobs=-1)
ensemble.fit(X_train, y_train)

# Evaluate tabular model (FIXED - Use consistent evaluation method)
y_pred = ensemble.predict(X_test)
test_score = r2_score(y_test, y_pred)  # Use r2_score for consistency
cv_scores = cross_val_score(ensemble, X_train, y_train, cv=8, scoring='r2', n_jobs=-1)

if abs(y_skewness) > 1:
    actual_prices = np.expm1(y_test)
    predicted_prices = np.expm1(y_pred)
else:
    actual_prices = y_test
    predicted_prices = y_pred

mae = mean_absolute_error(actual_prices, predicted_prices)

print(f"FIXED Tabular ensemble. R² = {test_score:.4f} (using r2_score), MAE = ${mae:.2f}")
```

Figure 7: Model Evaluation

```python
# 7. Model Performance Comparison (FIXED - Consistent Evaluation)
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

print("="*70)
print("FIXED MODEL PERFORMANCE COMPARISON")
print("="*70)
print("Note: Random Forest now uses regularization to prevent overfitting")
print("All models now use predict() + r2_score() for consistent evaluation")
print("="*70)

# Get predictions from all models (CONSISTENT METHOD)
lr_predictions = simple_lr.predict(X_test_scaled)
rf_predictions = simple_rf.predict(X_test_numerical)  # NOW USES REGULARIZED RF
ensemble_predictions = ensemble.predict(X_test)  # Use full X_test with all features
multimodal_predictions = multimodal_model.predict(X_test, text_data_test)

# Calculate comprehensive metrics
models_performance = {
    'Linear Regression': {
        'predictions': lr_predictions,
        'R²': r2_score(y_test, lr_predictions),
        'MAE': mean_absolute_error(y_test, lr_predictions),
        'RMSE': np.sqrt(mean_squared_error(y_test, lr_predictions)),
        'MAPE': np.mean(np.abs((y_test - lr_predictions) / y_test)) * 100
    },
    'Random Forest': {
        'predictions': rf_predictions,
        'R²': r2_score(y_test, rf_predictions),
        'MAE': mean_absolute_error(y_test, rf_predictions),
        'RMSE': np.sqrt(mean_squared_error(y_test, rf_predictions)),
        'MAPE': np.mean(np.abs((y_test - rf_predictions) / y_test)) * 100
    },
    'Ensemble': {
        'predictions': ensemble_predictions,
        'R²': r2_score(y_test, ensemble_predictions),
        'MAE': mean_absolute_error(y_test, ensemble_predictions),
        'RMSE': np.sqrt(mean_squared_error(y_test, ensemble_predictions)),
        'MAPE': np.mean(np.abs((y_test - ensemble_predictions) / y_test)) * 100
    },
    'Multimodal': {
        'predictions': multimodal_predictions,
        'R²': r2_score(y_test, multimodal_predictions),
        'MAE': mean_absolute_error(y_test, multimodal_predictions),
        'RMSE': np.sqrt(mean_squared_error(y_test, multimodal_predictions)),
        'MAPE': np.mean(np.abs((y_test - multimodal_predictions) / y_test)) * 100
    }
}

# Create performance comparison visualization
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Performance metrics comparison
metrics = ['R²', 'MAE', 'RMSE', 'MAPE']
model_names = list(models_performance.keys())

for i, metric in enumerate(metrics):
    values = [models_performance[model][metric] for model in model_names]

    row, col = i // 2, i % 2
    bars = axes[row, col].bar(model_names, values,
                              color=['skyblue', 'lightgreen', 'coral', 'gold'], alpha=0.8)
    axes[row, col].set_title(f'{metric} Comparison', fontsize=14, fontweight='bold')
    axes[row, col].set_ylabel(metric, fontsize=12)
    axes[row, col].grid(True, alpha=0.3)

    # Add value labels on bars
    for bar, value in zip(bars, values):
        height = bar.get_height()
        axes[row, col].text(bar.get_x() + bar.get_width()/2., height + height*0.01,
                            f'{value:.3f}', ha='center', va='bottom', fontweight='bold')

    # Rotate x-labels if needed
    if len(model_names[0]) > 8:
        axes[row, col].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.savefig('/Users/adityapandey/My Files/Thesis Sri Ganesh/Data Set/7/docs/images/model_performance_comparison.png',
            dpi=300, bbox_inches='tight')
plt.show()

# Print performance summary
print("Model Performance Summary:")
print("=" * 70)
for model, metrics in models_performance.items():
    print(f"\n{model}:")
    print(f"  R² Score:  {metrics['R²']:.4f}")
    print(f"  MAE:       ${metrics['MAE']:.2f}")
    print(f"  RMSE:      ${metrics['RMSE']:.2f}")
    print(f"  MAPE:      {metrics['MAPE']:.2f}%")
```

4.3 Explanations and Visualization

This section covers the generation of model explanations and visualizations.

Figure 8: SHAP: SHapley Additive exPlanations

```python
# Use the same preprocessing pipeline that was used for training
# Get numerical data only for SHAP analysis to avoid string conversion issues
sample_size = min(500, len(X_test))

# Use the preprocessed numerical data for SHAP
if 'X_test_numerical' in locals() and X_test_numerical is not None:
    X_shap_sample = X_test_numerical.sample(n=sample_size, random_state=42)
    print(f"Using numerical features: {X_shap_sample.shape[1]} features")
else:
    # Fallback: select only numerical columns from X_test
    numerical_columns = X_test.select_dtypes(include=[np.number]).columns
    X_shap_sample = X_test[numerical_columns].sample(n=sample_size, random_state=42)
    print(f"Using selected numerical features: {X_shap_sample.shape[1]} features")

# Ensure no NaN values
X_shap_sample = X_shap_sample.fillna(0)

# Create SHAP explainer for Random Forest (trained on numerical data)
explainer = shap.TreeExplainer(simple_rf)
shap_values = explainer.shap_values(X_shap_sample)

print(f"SHAP analysis completed for {sample_size} samples")

# Create comprehensive SHAP visualizations with improved formatting
fig = plt.figure(figsize=(24, 16))

# 1. SHAP Summary Plot (Bar) - Feature Importance Ranking
ax1 = plt.subplot(2, 2, 1)

# Calculate feature importance manually for better control
feature_importance = np.abs(shap_values).mean(0)
feature_names = X_shap_sample.columns
importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': feature_importance
}).sort_values('importance', ascending=False).head(15)
```

5 Application Demo

Streamlit Application is created below are screenshots of application. It will help AirBnB owners to understand factors affecting price and what is right price for property?

Figure 9: Home Page

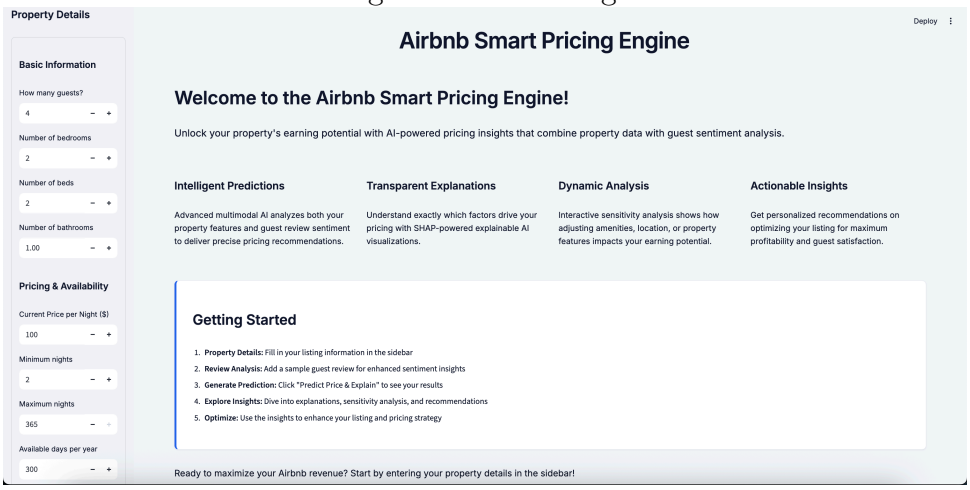

Figure 10: Predict and Explain Page:1

Figure 11: Predict and Explain Page:2