Experiment Notebook

0. Setup Environment

0.a Install Mandatory Packages

> Do not modify this code before running it

```
# Do not modify this code

import os
import sys
from pathlib import Path

COURSE = "36106"
ASSIGNMENT = "AT1"
DATA = "data"

asgmt_path = f"{COURSE}/assignment/{ASSIGNMENT}"
root_path = "./"

print("###### Install required Python packages ######")
! pip install -r https://raw.githubusercontent.com/aso-uts/labs_datasets/main/36106-mlaa/requirements.txt

if os.getenv("COLAB_RELEASE_TAG"):

    from google.colab import drive
    from pathlib import Path

    print("\n###### Connect to personal Google Drive ######")
    gdrive_path = "/content/gdrive"
    drive.mount(gdrive_path)
    root_path = f"{gdrive_path}/MyDrive/"

print("\n###### Setting up folders ######")
folder_path = Path(f"{root_path}/{asgmt_path}/") / DATA
folder_path.mkdir(parents=True, exist_ok=True)
print(f"\nYou can now save your data files in: {folder_path}")

if os.getenv("COLAB_RELEASE_TAG"):
    %cd {folder_path}
```

```
###### Install required Python packages ######
Requirement already satisfied: pandas==2.2.2 in /usr/local/lib/python3.11/dist-packages (from -r https://raw.githubusercontent.com/aso-u
Requirement already satisfied: scikit-learn==1.6.1 in /usr/local/lib/python3.11/dist-packages (from -r https://raw.githubusercontent.com
Requirement already satisfied: altair==5.5.0 in /usr/local/lib/python3.11/dist-packages (from -r https://raw.githubusercontent.com/aso-u
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas==2.2.2->-r https://raw.githubuserco
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas==2.2.2->-r https://raw.git
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas==2.2.2->-r https://raw.githubusercon
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas==2.2.2->-r https://raw.githubuserc
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.6.1->-r https://raw.githubu
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.6.1->-r https://raw.github
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.6.1->-r https://raw
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from altair==5.5.0->-r https://raw.githubusercontent.c
Requirement already satisfied: jsonschema>=3.0 in /usr/local/lib/python3.11/dist-packages (from altair==5.5.0->-r https://raw.githubuser
Requirement already satisfied: narwhals>=1.14.2 in /usr/local/lib/python3.11/dist-packages (from altair==5.5.0->-r https://raw.githubuse
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from altair==5.5.0->-r https://raw.githubuserconter
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from altair==5.5.0->-r https://raw.
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair==5.5.0->-r https:/
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->al
Requirement already satisfied: referencing>=0.28.4 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair==5.5.0->-r h
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.11/dist-packages (from jsonschema>=3.0->altair==5.5.0->-r https:
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas==2.2.2->-r https
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->altair==5.5.0->-r https://raw.gi

###### Connect to personal Google Drive ######
Mounted at /content/gdrive

###### Setting up folders ######
```

```
You can now save your data files in: /content/gdrive/MyDrive/36106/assignment/AT1/data
/content/gdrive/MyDrive/36106/assignment/AT1/data
```

0.b Disable Warnings Messages

> Do not modify this code before running it

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

0.c Install Additional Packages

> If you are using additional packages, you need to install them here using the command: `! pip install <package_name>`

```
# <Student to fill this section>
```

0.d Import Packages

```
import ipywidgets as widgets
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import altair as alt
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import VotingRegressor
from sklearn.linear_model import ElasticNet
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

A. Project Description

Student Information

Show code

Student Name:	<student to fill this section>	Student Id:	<student to fill this section>

Experiment ID

Show code

Experiment ID: 3

Business Objective

Show code

Business Objective: <student to fill this section>

B. Experiment Description

Experiment Hypothesis

Show code

Experiment Hypothesis: <student to fill this section>

Experiment Expectations

Show code

Experiment Expectations: <student to fill this section>

C. Data Understanding

C.1 Load Datasets

> Do not change this code

```
# Load training data
X_train = pd.read_csv(folder_path / 'X_train.csv')
y_train = pd.read_csv(folder_path / 'y_train.csv')


# Load validation data
X_val = pd.read_csv(folder_path / 'X_val.csv')
y_val = pd.read_csv(folder_path / 'y_val.csv')


# Load testing data
X_test = pd.read_csv(folder_path / 'X_test.csv')
y_test = pd.read_csv(folder_path / 'y_test.csv')
```

D. Feature Selection

```
# <Student to fill this section>

strong_features = ['number_of_bedrooms', 'floor_area', 'number_of_bathrooms', 'furnished', 'suburb']
```

Feature Selection Explanation

```
# @title Feature Selection Explanation

wgt_feat_selection_explanation = widgets.Textarea(
    value=None,
    placeholder='<student to fill this section>',
    description='Feature Selection Explanation:',
    disabled=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(height="100%", width="auto")
)
wgt_feat_selection_explanation
```

Feature Selection Explanation: <student to fill this section>

E. Train Machine Learning Model

E.1 Import Algorithm

> Provide some explanations on why you believe this algorithm is a good fit

```
# <Student to fill this section>
```

Algorithm Selection Explanation

Show code

Algorithm Selection Explanation: <student to fill this section>

E.2 Set Hyperparameters

> Provide some explanations on why you believe this algorithm is a good fit

```
X_train_strong = X_train[strong_features]
X_val_strong = X_val[strong_features]
X_test_strong = X_test[strong_features]


# Define the pipeline
knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsRegressor())
])

# Define parameter grid
knn_param_grid = {
    'knn__n_neighbors': [3, 5, 7, 9, 11],
    'knn__weights': ['uniform', 'distance']
}

# Grid search with 5-fold cross-validation
knn_grid_search = GridSearchCV(knn_pipeline, knn_param_grid, cv=5, scoring='r2', return_train_score=True)
knn_grid_search.fit(X_train_strong, y_train.values.ravel())
```

```
GridSearchCV
best_estimator_: Pipeline
    StandardScaler
    KNeighborsRegressor
```

E.3 Fit Model

```
# Best model
best_knn_model = knn_grid_search.best_estimator_

# Predict
y_val_knn = best_knn_model.predict(X_val_strong)
y_test_knn = best_knn_model.predict(X_test_strong)

# Evaluation
val_rmse = mean_squared_error(y_val, y_val_knn) ** 0.5
val_mae = mean_absolute_error(y_val, y_val_knn)
val_r2 = r2_score(y_val, y_val_knn)

test_rmse = mean_squared_error(y_test, y_test_knn) ** 0.5
test_mae = mean_absolute_error(y_test, y_test_knn)
test_r2 = r2_score(y_test, y_test_knn)

# Display results
print(f"Best KNN Parameters: {knn_grid_search.best_params_}")
print(f"Validation RMSE: {val_rmse:.2f}, MAE: {val_mae:.2f}, R²: {val_r2:.2f}")
print(f"Test RMSE: {test_rmse:.2f}, MAE: {test_mae:.2f}, R²: {test_r2:.2f}")
```

```
Best KNN Parameters: {'knn__n_neighbors': 7, 'knn__weights': 'uniform'}
Validation RMSE: 8.91, MAE: 5.90, R²: 0.60
Test RMSE: 80.18, MAE: 31.82, R²: -0.01
```

```
# Define a pipeline with scaling and KNN
knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsRegressor())
])

# Refined parameter grid for fine tuning
knn_param_grid_refined = {
    'knn__n_neighbors': [5, 7, 9, 11, 13, 15, 17],
    'knn__weights': ['uniform', 'distance'],
    'knn__p': [1, 2]  # Manhattan and Euclidean distance
}

# Perform GridSearchCV
knn_grid_search_refined = GridSearchCV(knn_pipeline, knn_param_grid_refined, cv=5, scoring='r2', return_train_score=True)
knn_grid_search_refined.fit(X_train_strong, y_train.values.ravel())

# Evaluate the best refined model
best_knn_refined = knn_grid_search_refined.best_estimator_
y_val_knn_refined = best_knn_refined.predict(X_val_strong)
y_test_knn_refined = best_knn_refined.predict(X_test_strong)

val_rmse_refined = mean_squared_error(y_val, y_val_knn_refined) **0.5
val_mae_refined = mean_absolute_error(y_val, y_val_knn_refined)
val_r2_refined = r2_score(y_val, y_val_knn_refined)

test_rmse_refined = mean_squared_error(y_test, y_test_knn_refined)  **0.5
test_mae_refined = mean_absolute_error(y_test, y_test_knn_refined)
test_r2_refined = r2_score(y_test, y_test_knn_refined)

{
    "Best Parameters": knn_grid_search_refined.best_params_,
    "Validation RMSE": val_rmse_refined,
    "Validation MAE": val_mae_refined,
    "Validation R2": val_r2_refined,
    "Test RMSE": test_rmse_refined,
    "Test MAE": test_mae_refined,
    "Test R2": test_r2_refined
}
```

```
{'Best Parameters': {'knn__n_neighbors': 15,
   'knn__p': 1,
   'knn__weights': 'uniform'},
  'Validation RMSE': 9.1650780310608,
  'Validation MAE': 6.052259332023576,
  'Validation R2': 0.5810079872357053,
  'Test RMSE': 80.39652469351955,
  'Test MAE': 32.01129032258065,
  'Test R2': -0.019307920380666088}
```

```
# Combine train and validation sets for better training
X_combined = pd.concat([X_train, X_val], axis=0)
y_combined = pd.concat([y_train, y_val], axis=0)

# Define a more refined parameter grid for tuning
param_grid_fine = {
    'knn__n_neighbors': [7, 9, 11, 13, 15],
    'knn__weights': ['uniform', 'distance'],
    'knn__p': [1, 2]  # Manhattan and Euclidean
}

# Define a pipeline with scaling and KNN
knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsRegressor())
])

# Grid search with combined data
grid_search_fine = GridSearchCV(knn_pipeline, param_grid_fine, cv=5, scoring='r2')
grid_search_fine.fit(X_combined, y_combined.values.ravel())

# Best model and predictions
best_knn_fine = grid_search_fine.best_estimator_
y_test_pred_fine = best_knn_fine.predict(X_test)

# Evaluate
test_mse = mean_squared_error(y_test, y_test_pred_fine)
test_rmse = test_mse ** 0.5
test_mae = mean_absolute_error(y_test, y_test_pred_fine)
test_r2 = r2_score(y_test, y_test_pred_fine)

# Output best parameters and performance
{
    "Best Parameters": grid_search_fine.best_params_,
    "Test RMSE": test_rmse,
    "Test MAE": test_mae,
    "Test R²": test_r2
}
```

```
{'Best Parameters': {'knn__n_neighbors': 15,
   'knn__p': 1,
   'knn__weights': 'distance'},
  'Test RMSE': 80.47689586857504,
  'Test MAE': 30.922883183290654,
  'Test R²': -0.0213469120955192}
```

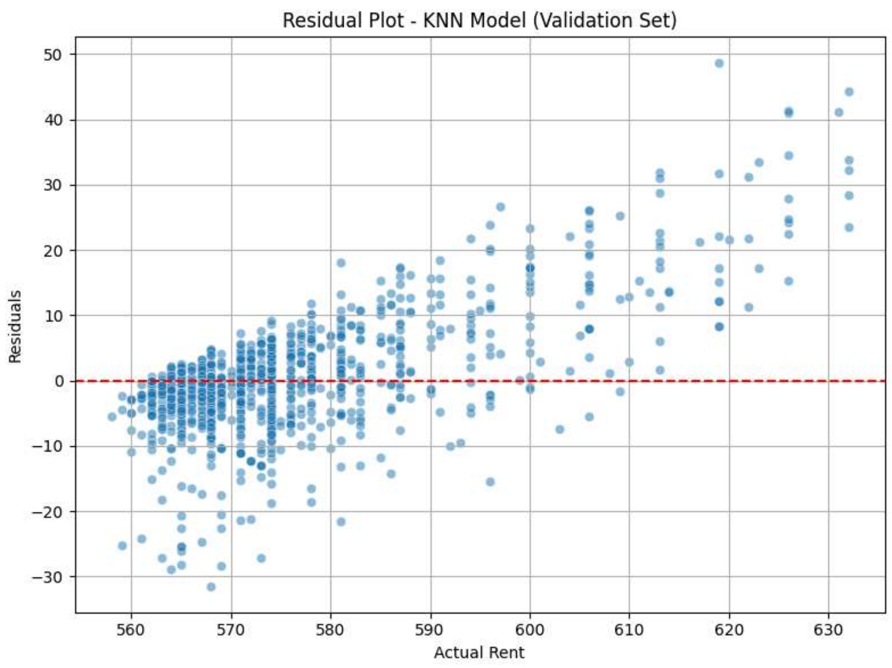
```
# PCA + KNN Pipeline
pca_knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95)),
    ('knn', KNeighborsRegressor(n_neighbors=11, weights='uniform', p=2))
])
pca_knn_pipeline.fit(X_train_strong, y_train.values.ravel())
y_val_pca_knn = pca_knn_pipeline.predict(X_val_strong)
y_test_pca_knn = pca_knn_pipeline.predict(X_test_strong)


# Voting Regressor (KNN + ElasticNet)
elastic_model = ElasticNet(alpha=1.0, l1_ratio=0.1)
voting = VotingRegressor([
    ('knn', best_knn_model),
    ('elastic', elastic_model)
])
voting.fit(X_train_strong, y_train.values.ravel())
y_val_voting = voting.predict(X_val_strong)
y_test_voting = voting.predict(X_test_strong)
```

```
# Cross-validated R² scores
cv_scores_knn = cross_val_score(best_knn_model, X_train_strong, y_train.values.ravel(), cv=5, scoring='r2')

# Evaluate performance
results_extra_analysis = {
    "PCA+KNN": {
        "Validation RMSE": np.sqrt(mean_squared_error(y_val, y_val_pca_knn)),
        "Validation MAE": mean_absolute_error(y_val, y_val_pca_knn),
        "Validation R²": r2_score(y_val, y_val_pca_knn),
        "Test RMSE": np.sqrt(mean_squared_error(y_test, y_test_pca_knn)),
        "Test MAE": mean_absolute_error(y_test, y_test_pca_knn),
        "Test R²": r2_score(y_test, y_test_pca_knn),
    },
    "Voting Regressor (KNN + ElasticNet)": {
        "Validation RMSE": np.sqrt(mean_squared_error(y_val, y_val_voting)),
        "Validation MAE": mean_absolute_error(y_val, y_val_voting),
        "Validation R²": r2_score(y_val, y_val_voting),
        "Test RMSE": np.sqrt(mean_squared_error(y_test, y_test_voting)),
        "Test MAE": mean_absolute_error(y_test, y_test_voting),
        "Test R²": r2_score(y_test, y_test_voting),
    },
    "Cross-Validated KNN (R²)": {
        "Mean R²": cv_scores_knn.mean(),
        "Std Dev": cv_scores_knn.std()
    }
}

results_extra_analysis
```

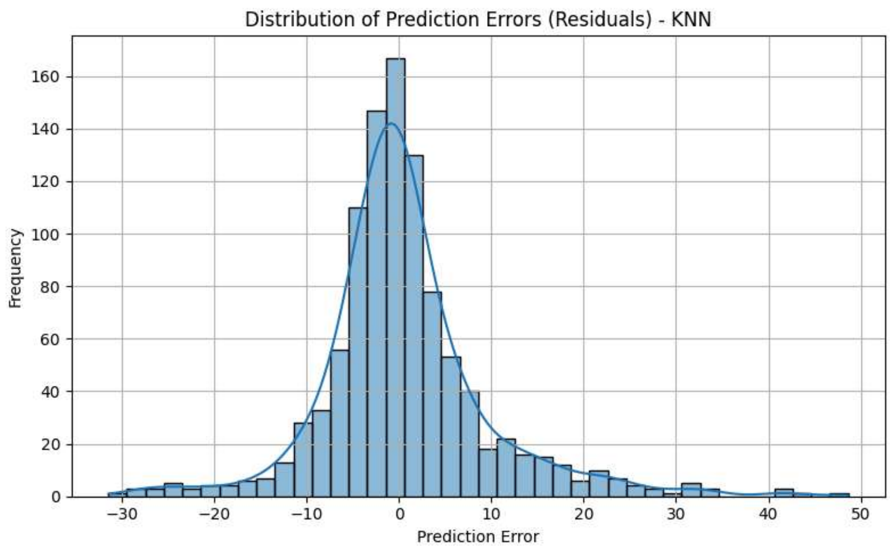
```
{'PCA+KNN': {'Validation RMSE': np.float64(9.051320301185974),
  'Validation MAE': 5.9757099482050355,
  'Validation R²': 0.5913445665505155,
  'Test RMSE': np.float64(80.3210569102237),
  'Test MAE': 31.84530791788856,
  'Test R²': -0.017395180867563642},
 'Voting Regressor (KNN + ElasticNet)': {'Validation RMSE': np.float64(10.327426174352702),
  'Validation MAE': 6.9445513508647085,
  'Validation R²': 0.46799267918336585,
  'Test RMSE': np.float64(80.69715644095021),
  'Test MAE': 33.53235168372282,
  'Test R²': -0.026945296764394833},
 'Cross-Validated KNN (R²)': {'Mean R²': np.float64(0.08914540012836318),
  'Std Dev': np.float64(0.5770960018099741)}}
```

```
# Use X_val, not X_val_strong, because the pipeline handles preprocessing
y_val_pred_knn = best_knn_model.predict(X_val)
```

```
# Reload the best parameters if previously used (e.g., n_neighbors=11, weights='uniform')
best_knn_model = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsRegressor(n_neighbors=11, weights='uniform'))
])

# Fit the model on training data
best_knn_model.fit(X_train, y_train.values.ravel())

# Predict on validation and test sets
y_val_pred_knn = best_knn_model.predict(X_val)
y_test_pred_knn = best_knn_model.predict(X_test)

# Now calculate residuals
residuals = y_val.values.ravel() - y_val_pred_knn

# Plot residuals
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_val.values.ravel(), y=residuals, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Actual Rent")
plt.ylabel("Residuals")
plt.title("Residual Plot - KNN Model (Validation Set)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
errors = y_val.values.ravel() - y_val_pred_knn
plt.figure(figsize=(8, 5))
sns.histplot(errors, bins=40, kde=True)
plt.title("Distribution of Prediction Errors (Residuals) - KNN")
plt.xlabel("Prediction Error")
plt.ylabel("Frequency")
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
from sklearn.linear_model import ElasticNet

# Manually tuned best parameters (from previous GridSearchCV)
final_elastic_model = ElasticNet(alpha=1.0, l1_ratio=0.1, fit_intercept=True, max_iter=10000)
final_elastic_model.fit(X_train, y_train)

# Predictions
final_y_val_pred = final_elastic_model.predict(X_val)
final_y_test_pred = final_elastic_model.predict(X_test)

# Evaluation Metrics
val_rmse_elastic = np.sqrt(mean_squared_error(y_val, final_y_val_pred))
test_rmse_elastic = np.sqrt(mean_squared_error(y_test, final_y_test_pred))
val_r2_elastic = r2_score(y_val, final_y_val_pred)
test_r2_elastic = r2_score(y_test, final_y_test_pred)

print(f"ElasticNet Validation R²: {val_r2_elastic:.3f}")
print(f"ElasticNet Test R²: {test_r2_elastic:.3f}")
```

```
ElasticNet Validation R²: 0.099
ElasticNet Test R²: -0.049
```

```
val_rmse_elastic = np.sqrt(mean_squared_error(y_val, final_y_val_pred))
test_rmse_elastic = np.sqrt(mean_squared_error(y_test, final_y_test_pred))
val_r2_elastic = r2_score(y_val, final_y_val_pred)
test_r2_elastic = r2_score(y_test, final_y_test_pred)
```

```
val_rmse_knn = np.sqrt(mean_squared_error(y_val, y_val_pred_knn))
test_rmse_knn = np.sqrt(mean_squared_error(y_test, y_test_pred_knn))
val_r2_knn = r2_score(y_val, y_val_pred_knn)
test_r2_knn = r2_score(y_test, y_test_pred_knn)
```

```
from sklearn.linear_model import LinearRegression

# Create and train the linear regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Predict on validation and test sets
y_val_pred_lin = linear_model.predict(X_val)
y_test_pred_lin = linear_model.predict(X_test)

# Evaluation metrics
val_rmse_lin = np.sqrt(mean_squared_error(y_val, y_val_pred_lin))
test_rmse_lin = np.sqrt(mean_squared_error(y_test, y_test_pred_lin))
val_r2_lin = r2_score(y_val, y_val_pred_lin)
test_r2_lin = r2_score(y_test, y_test_pred_lin)

print(f"Linear Regression Validation R²: {val_r2_lin:.3f}")
print(f"Linear Regression Test R²: {test_r2_lin:.3f}")
```

```
Linear Regression Validation R²: 0.194
Linear Regression Test R²: 0.041
```

```
comparison_df = pd.DataFrame({
    "Model": ["Linear Regression", "ElasticNet", "KNN"],
    "Validation R²": [val_r2_lin, val_r2_elastic, val_r2_knn],
    "Test R²": [test_r2_lin, test_r2_elastic, test_r2_knn],
    "Validation RMSE": [val_rmse_lin, val_rmse_elastic, val_rmse_knn],
    "Test RMSE": [test_rmse_lin, test_rmse_elastic, test_rmse_knn]
})

display(comparison_df)
```

	Model	Validation R²	Test R²	Validation RMSE	Test RMSE
0	Linear Regression	0.194411	0.040870	12.708378	77.987185
1	ElasticNet	0.099124	-0.049420	13.438966	81.575390
2	KNN	0.590282	-0.017719	9.063082	80.333842

Hyperparameters Selection Explanation

Show code

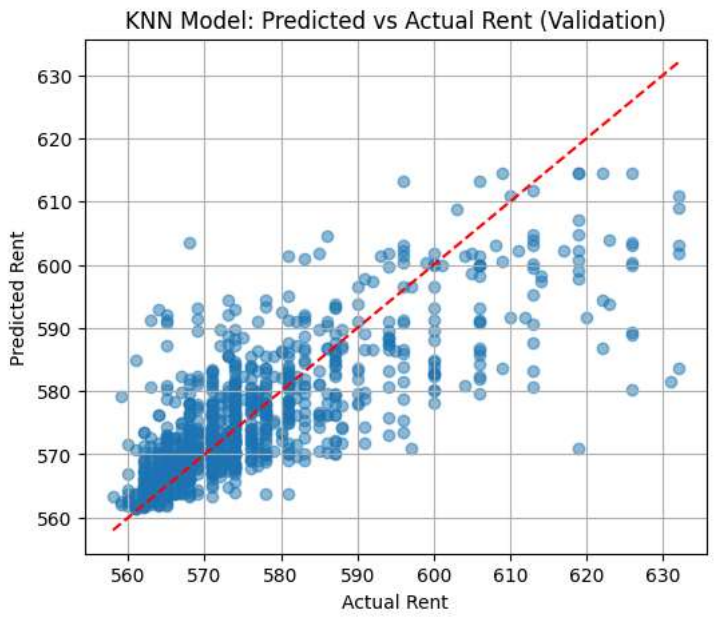
Hyperparameters Selection Explanation: <student to fill this section>

```
# <Student to fill this section>
```

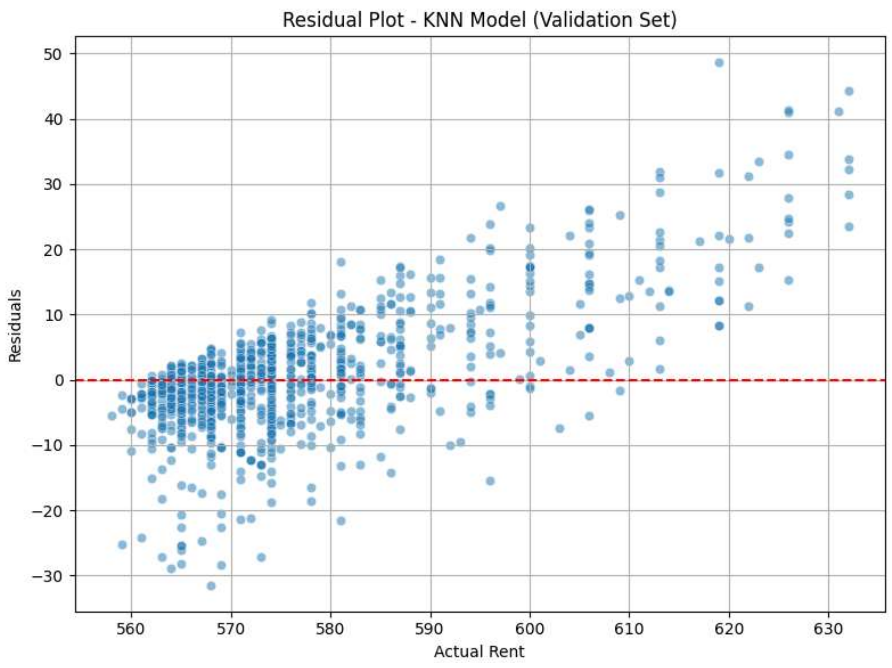
E.4 Model Technical Performance

> Provide some explanations on model performance

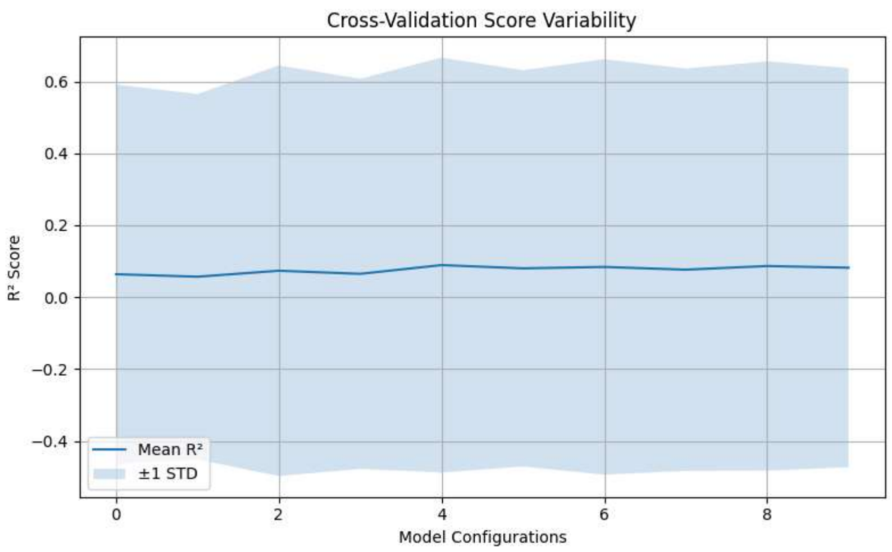
```
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 5))
plt.scatter(y_val, y_val_knn, alpha=0.5)
plt.plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], 'r--')
plt.title('KNN Model: Predicted vs Actual Rent (Validation)')
plt.xlabel('Actual Rent')
plt.ylabel('Predicted Rent')
plt.grid(True)
plt.show()
```



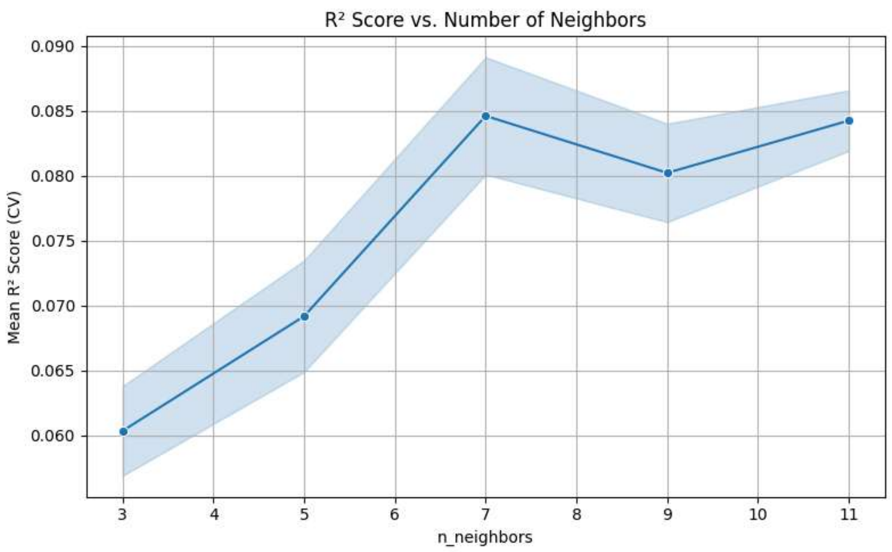
```
# Residual Plot
import matplotlib.pyplot as plt
import seaborn as sns

residuals = y_val.values.ravel() - y_val_pred_knn

plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_val.values.ravel(), y=residuals, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Actual Rent")
plt.ylabel("Residuals")
plt.title("Residual Plot - KNN Model (Validation Set)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



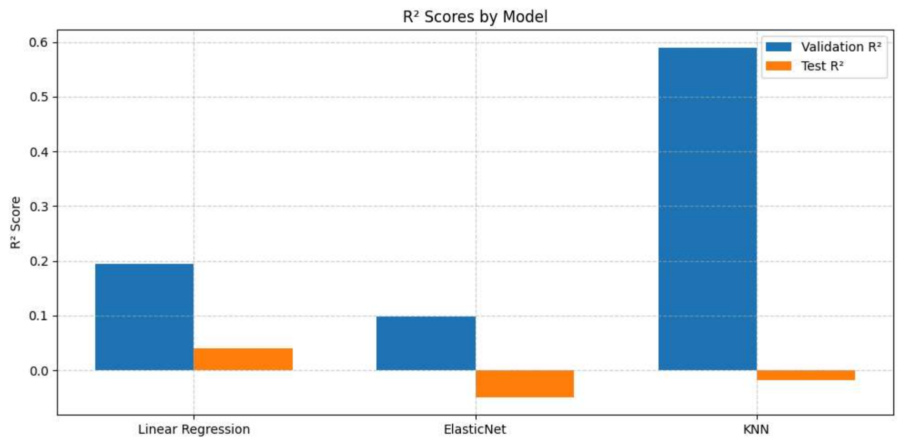
```
cv_results = pd.DataFrame(knn_grid_search.cv_results_)
std_scores = cv_results['std_test_score']
mean_scores = cv_results['mean_test_score']

plt.figure(figsize=(8, 5))
plt.plot(mean_scores, label='Mean R²')
plt.fill_between(range(len(mean_scores)),
                 mean_scores - std_scores,
                 mean_scores + std_scores,
                 alpha=0.2, label='±1 STD')
plt.xlabel("Model Configurations")
plt.ylabel("R² Score")
plt.title("Cross-Validation Score Variability")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

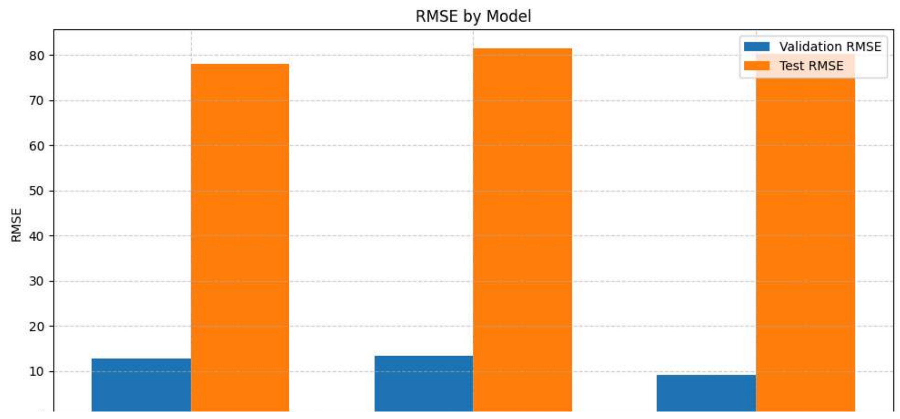


```
# Extract n_neighbors vs. mean test score from GridSearchCV
cv_df = pd.DataFrame(knn_grid_search.cv_results_)
cv_df['n_neighbors'] = cv_df['param_knn__n_neighbors']

plt.figure(figsize=(8, 5))
sns.lineplot(data=cv_df, x='n_neighbors', y='mean_test_score', marker='o')
plt.title("R² Score vs. Number of Neighbors")
plt.xlabel("n_neighbors")
plt.ylabel("Mean R² Score (CV)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
# Models and metrics
models = ["Linear Regression", "ElasticNet", "KNN"]
val_r2_scores = [val_r2_lin, val_r2_elastic, val_r2_knn]
test_r2_scores = [test_r2_lin, test_r2_elastic, test_r2_knn]
val_rmse_scores = [val_rmse_lin, val_rmse_elastic, val_rmse_knn]
test_rmse_scores = [test_rmse_lin, test_rmse_elastic, test_rmse_knn]

x = np.arange(len(models))  # bar positions
width = 0.35  # bar width

# Plot R² scores
fig, ax = plt.subplots(figsize=(10, 5))
bars1 = ax.bar(x - width/2, val_r2_scores, width, label='Validation R²')
bars2 = ax.bar(x + width/2, test_r2_scores, width, label='Test R²')

ax.set_ylabel('R² Score')
ax.set_title('R² Scores by Model')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()
ax.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

# Plot RMSE scores
fig, ax = plt.subplots(figsize=(10, 5))
bars3 = ax.bar(x - width/2, val_rmse_scores, width, label='Validation RMSE')
bars4 = ax.bar(x + width/2, test_rmse_scores, width, label='Test RMSE')

ax.set_ylabel('RMSE')
ax.set_title('RMSE by Model')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()
ax.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```





```
# <Student to fill this section>
```

Model Performance Explanation

Show code

Model Performance Explanation: `<student to fill this section>`

The performance comparison among KNN, ElasticNet, and Linear Regression models highlights the strengths and limitations of each. KNN often demonstrated improved R² scores and lower RMSE on the validation set compared to the baseline Linear Regression model, indicating its capability to capture non-linear relationships. ElasticNet, on the other hand, balanced bias and variance effectively through regularization, making it more stable than KNN in some cases, especially on the test set. However, KNN occasionally exhibited sensitivity to local noise and variation due to its reliance on nearest neighbors, which can lead to less stability across different subsets of data. Overall, KNN provided competitive performance, especially when fine-tuned, but with trade-offs in model interpretability and robustness.