Name: Abdul Rafay

BS Software Engineering (4th)

Arch Technologies Category B – Task 2

Chapter 1: Machine Learning Landscape

➢ What is Machine Learning?

Machine Learning is all about teaching computers to learn from data and make decisions without being specifically programmed for every task. A helpful way to think about it is through a definition by Tom Mitchell:

A computer learns when it gets better at a task (T), using experience (E), and its performance improves over time, as measured by some criteria (P).

➢ Types of Machine Learning

- **Supervised Learning**: The model learns using labeled data, meaning we already know the answers. It's used in tasks like predicting prices (regression) or identifying objects (classification).
- **Unsupervised Learning**: Here, the data has no labels. The goal is to find patterns or groupings. Examples include clustering and dimensionality reduction (like PCA).
- **Semi-supervised Learning**: A mix of both labeled and unlabeled data. This is useful when labeling data is expensive or time-consuming.
- **Reinforcement Learning**: The model learns by interacting with an environment and receiving feedback, like rewards or penalties, much like training a pet.

➢ How Models Learn

- **Batch Learning**: The system is trained all at once using all the available data.
- **Online Learning**: The model learns little by little, as new data comes in. This is useful for real-time systems.

➢ Learning Techniques

- **Instance-Based Learning**: This method works by comparing new problems with past examples, kind of like looking up similar cases. An example is k-Nearest Neighbors.

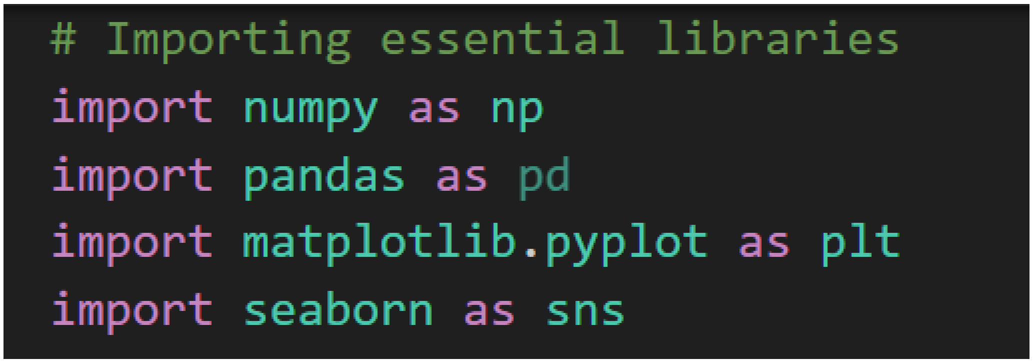
- **Model-Based Learning**: In this case, the system tries to build a general model from the training data. Linear regression is a good example.

➢ Common Challenges

Some typical issues in machine learning include not having enough data, poor-quality data, overfitting (model learns too much from the training data and doesn't generalize), underfitting (model is too simple), and tuning hyperparameters to get better results.

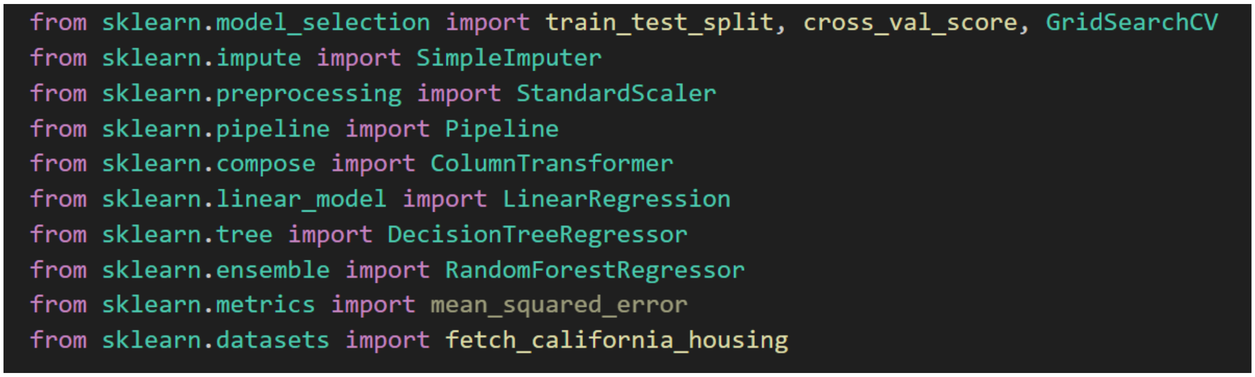
Chapter 2: End to End Machine Learning Project

➢ Important Libraries:



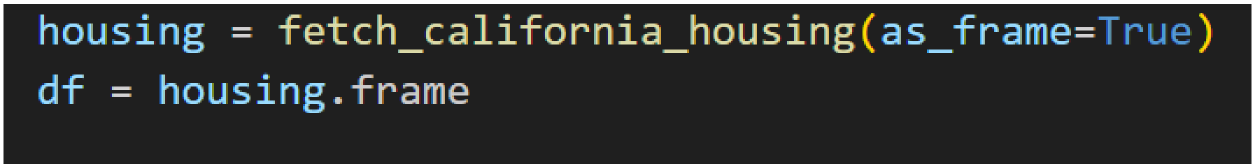
```
# Importing essential libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

➢ Scikit-Learn Tools:



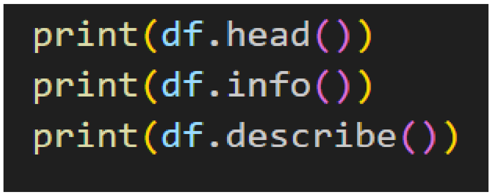
```
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.datasets import fetch_california_housing
```

- ➢ Dataset:



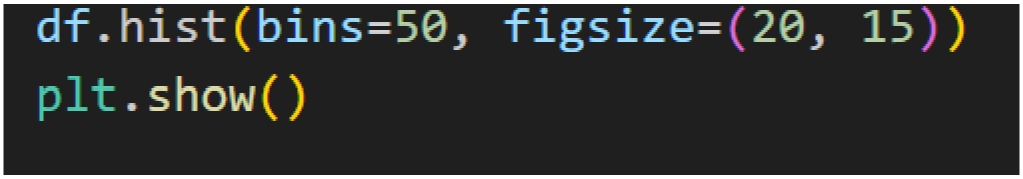
```
housing = fetch_california_housing(as_frame=True)
df = housing.frame
```

- ➢ Initial Data Check:



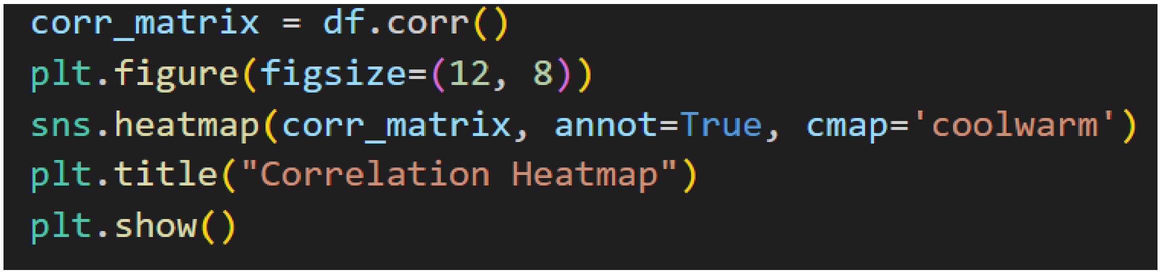
```
print(df.head())
print(df.info())
print(df.describe())
```

- ➢ Histogram:



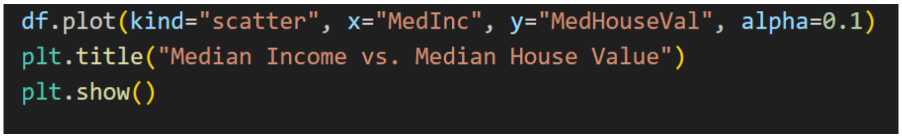
```
df.hist(bins=50, figsize=(20, 15))
plt.show()
```

- ➢ Correlation matrix heatmap:



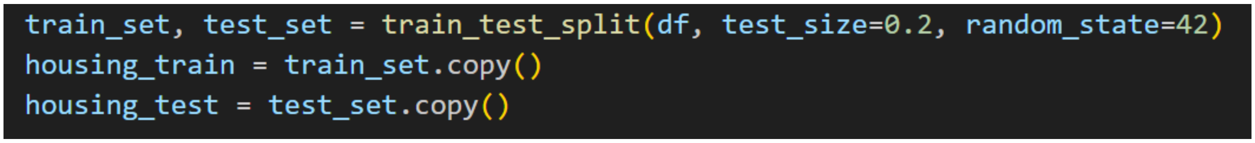
```
corr_matrix = df.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

- ➢ Scatterplot:



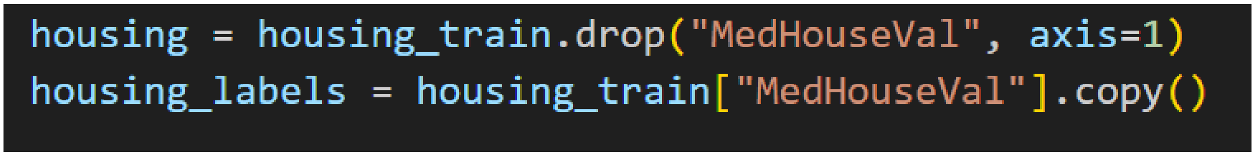
```
df.plot(kind="scatter", x="MedInc", y="MedHouseVal", alpha=0.1)
plt.title("Median Income vs. Median House Value")
plt.show()
```

➢ Splitting the data:



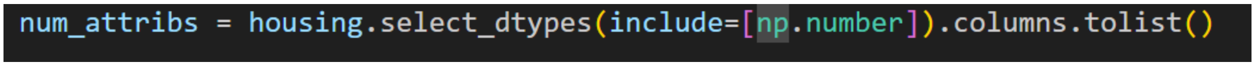
```python
train_set, test_set = train_test_split(df, test_size=0.2, random_state=42)
housing_train = train_set.copy()
housing_test = test_set.copy()
```

➢ Separating features and labels:



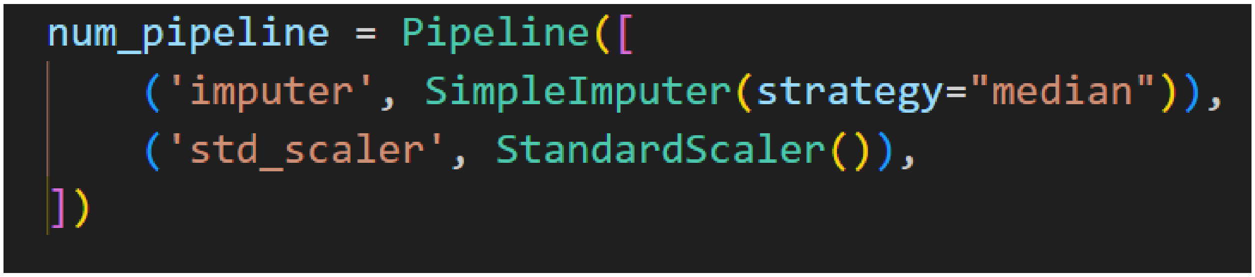
```python
housing = housing_train.drop("MedHouseVal", axis=1)
housing_labels = housing_train["MedHouseVal"].copy()
```

➢ Numerical Features:



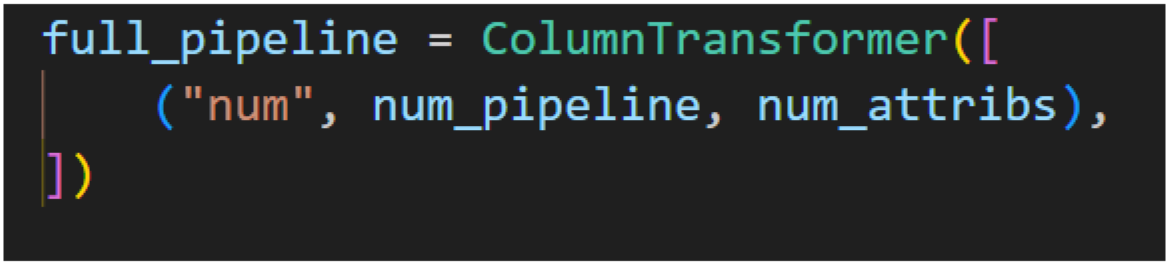
```python
num_attribs = housing.select_dtypes(include=[np.number]).columns.tolist()
```

➢ Pipeline for numerical data preprocessing:



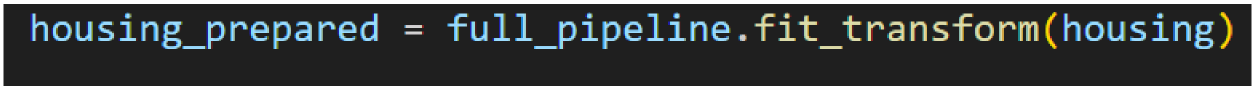
```python
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler()),
])
```

➢ Full preprocessing pipeline:



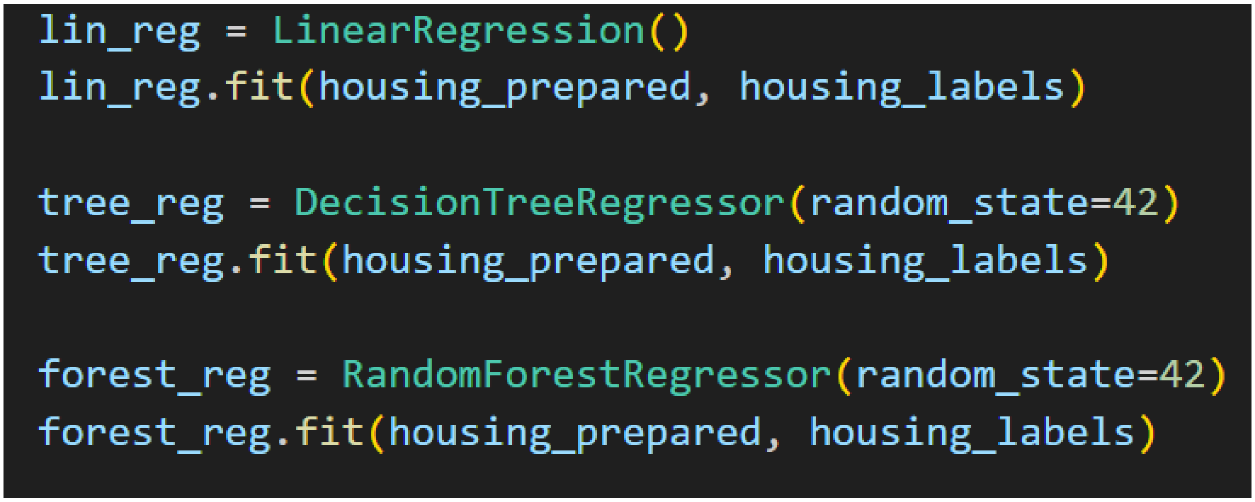
```python
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
])
```

➢ Preparing data:



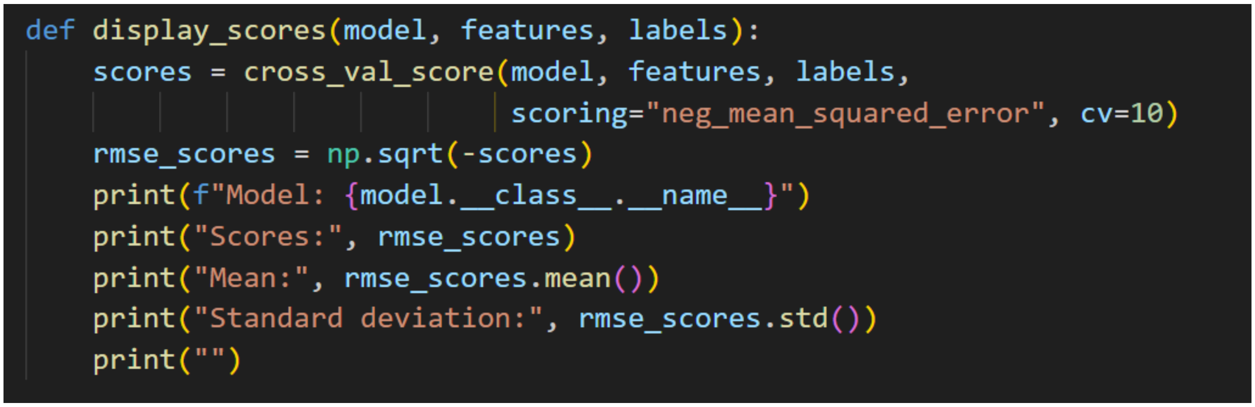
```
housing_prepared = full_pipeline.fit_transform(housing)
```

➢ Models trained:



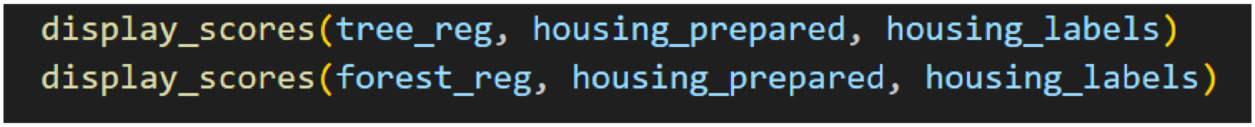
```
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)

forest_reg = RandomForestRegressor(random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

➢ Cross Validation RMSE Scores:



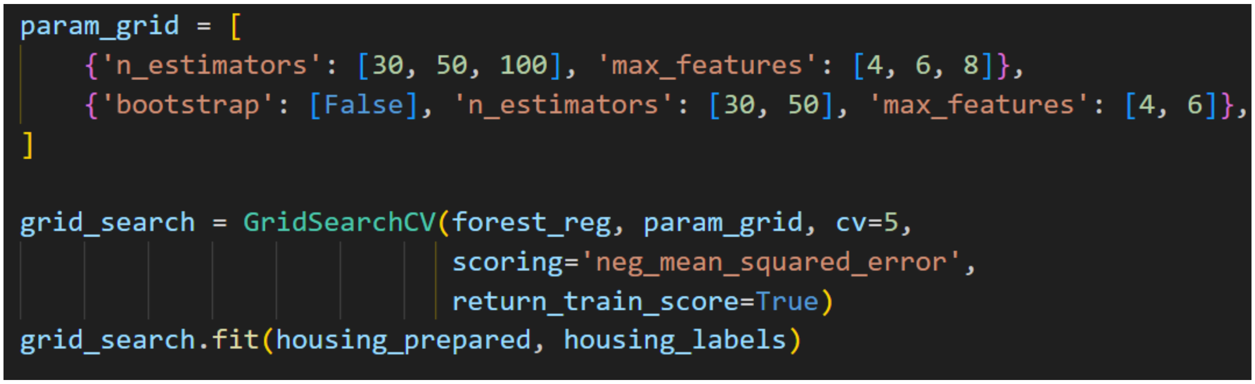
```
def display_scores(model, features, labels):
    scores = cross_val_score(model, features, labels,
                             scoring="neg_mean_squared_error", cv=10)
    rmse_scores = np.sqrt(-scores)
    print(f"Model: {model.__class__.__name__}")
    print("Scores:", rmse_scores)
    print("Mean:", rmse_scores.mean())
    print("Standard deviation:", rmse_scores.std())
    print("")
```

➢ Evaluating Models:



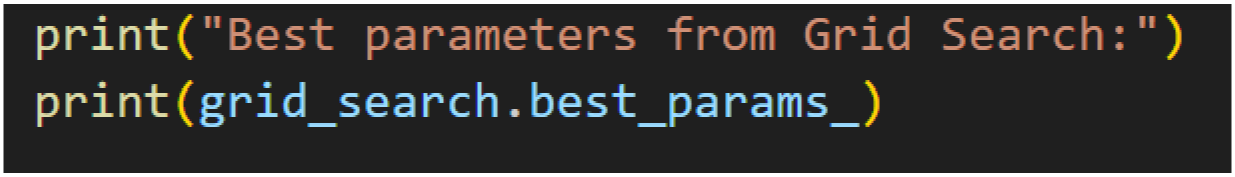
```
display_scores(tree_reg, housing_prepared, housing_labels)
display_scores(forest_reg, housing_prepared, housing_labels)
```

- Grid Search for best hyperparameters for Random Forest:



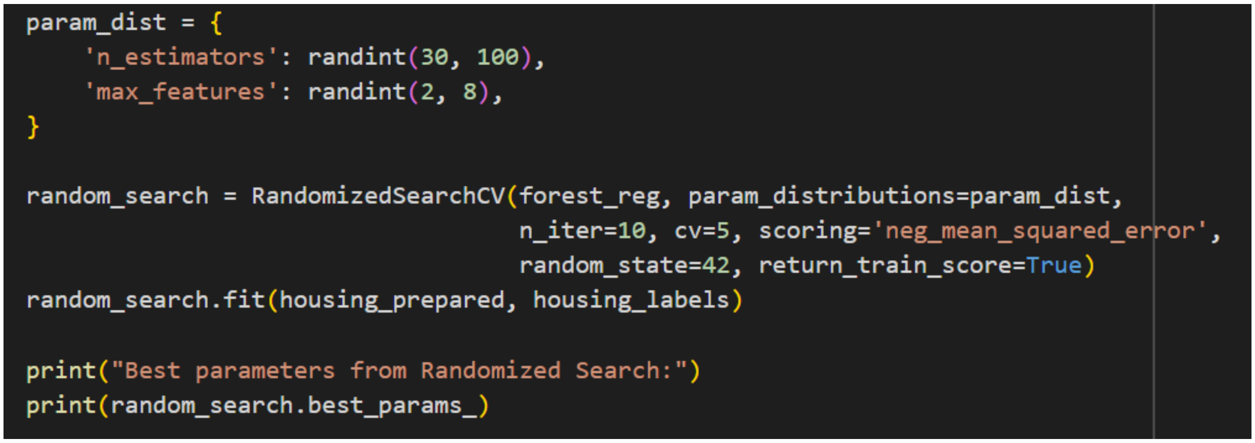
```
param_grid = [
    {'n_estimators': [30, 50, 100], 'max_features': [4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [30, 50], 'max_features': [4, 6]},
]

grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

- Output best parameter:



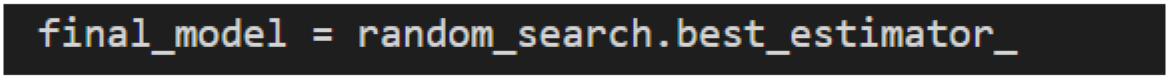
```
print("Best parameters from Grid Search:")
print(grid_search.best_params_)
```

- Randomized Search:



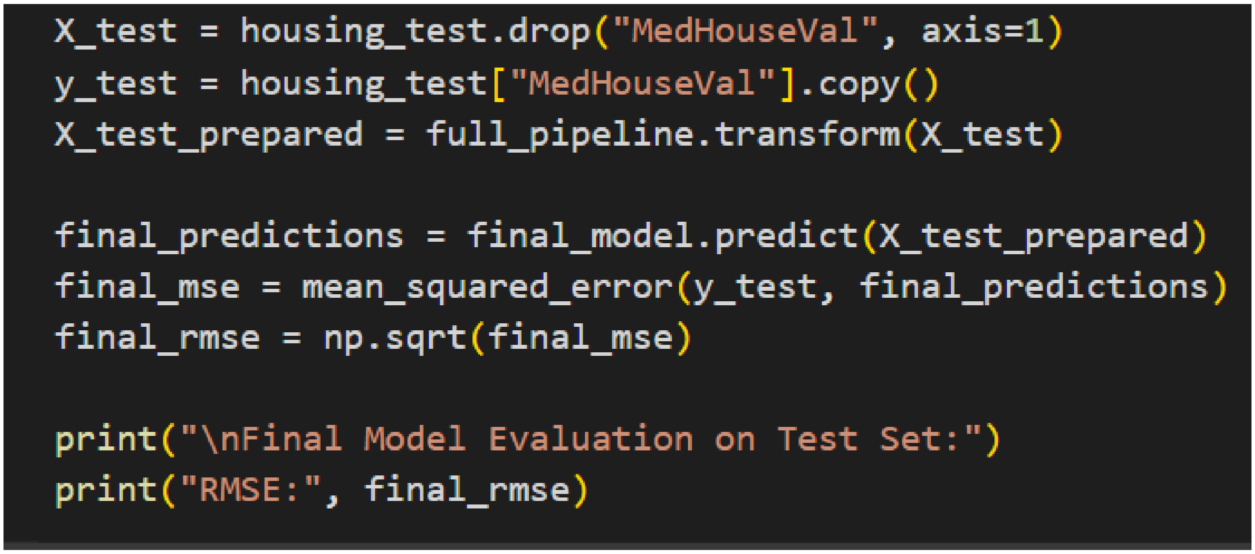
```
param_dist = {
    'n_estimators': randint(30, 100),
    'max_features': randint(2, 8),
}

random_search = RandomizedSearchCV(forest_reg, param_distributions=param_dist,
                                   n_iter=10, cv=5, scoring='neg_mean_squared_error',
                                   random_state=42, return_train_score=True)
random_search.fit(housing_prepared, housing_labels)

print("Best parameters from Randomized Search:")
print(random_search.best_params_)
```

- Evaluate Final Model



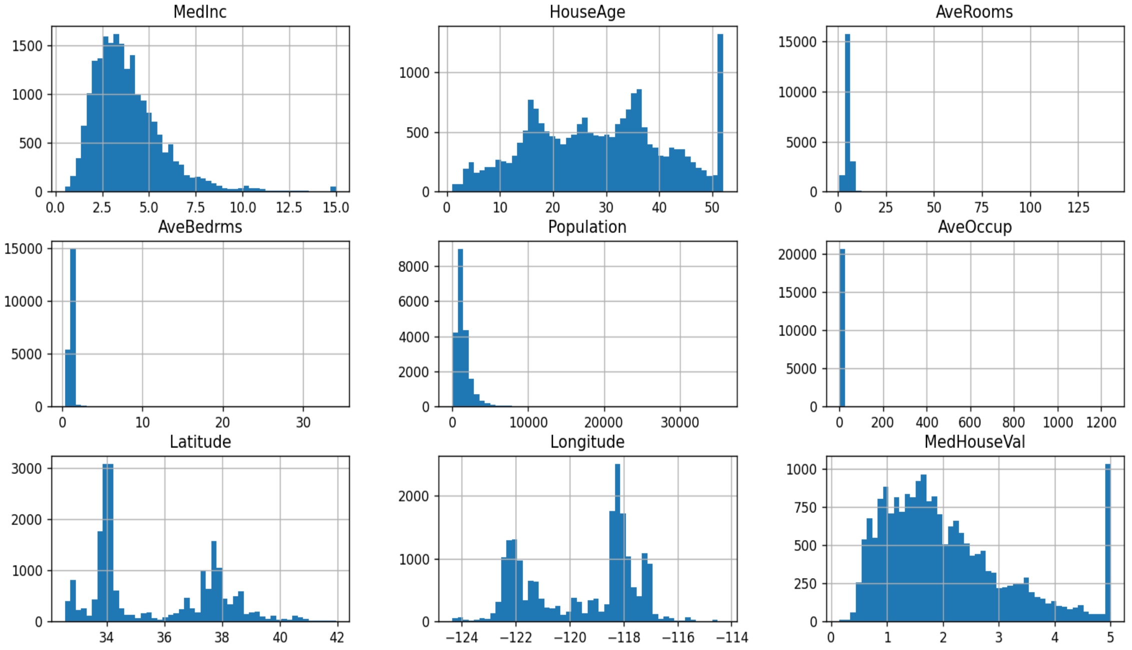
```
final_model = random_search.best_estimator_
```

➢ Prepare for the test set:

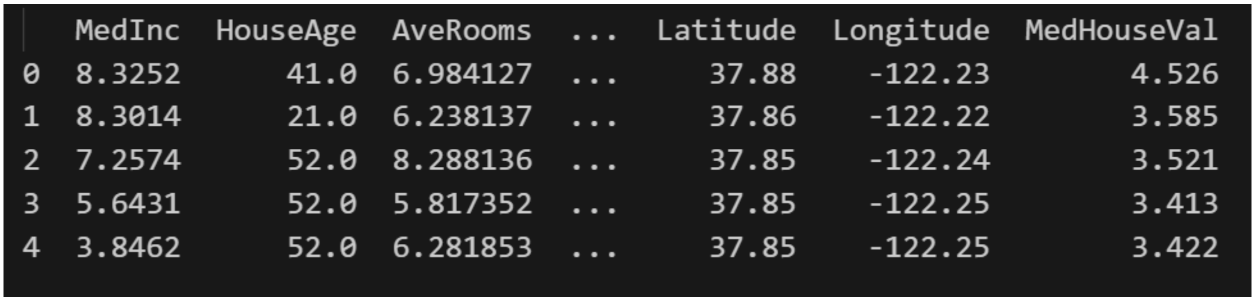


```
X_test = housing_test.drop("MedHouseVal", axis=1)
y_test = housing_test["MedHouseVal"].copy()
X_test_prepared = full_pipeline.transform(X_test)

final_predictions = final_model.predict(X_test_prepared)
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)

print("\nFinal Model Evaluation on Test Set:")
print("RMSE:", final_rmse)
```

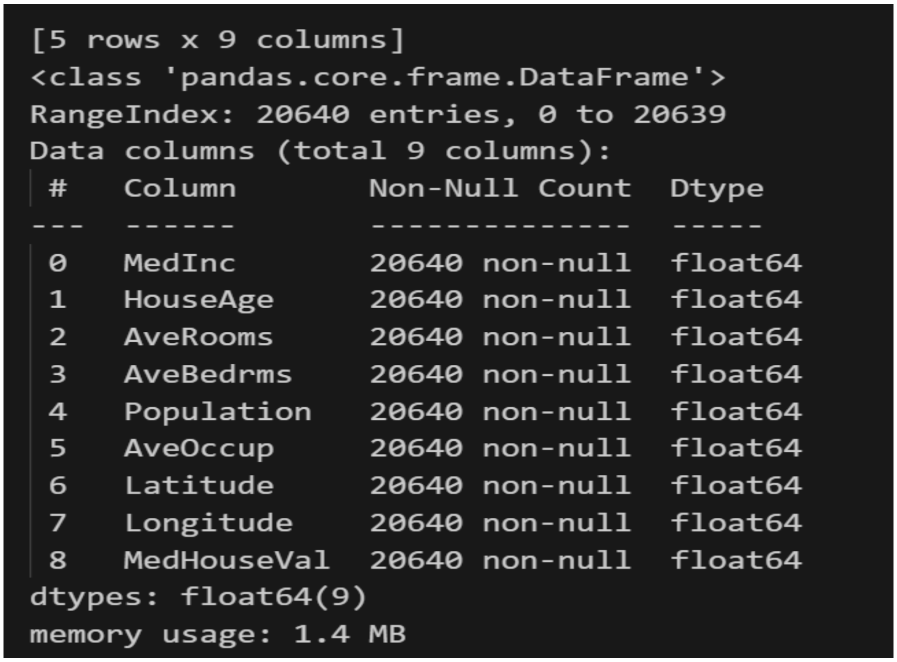
➢ All Graphs:



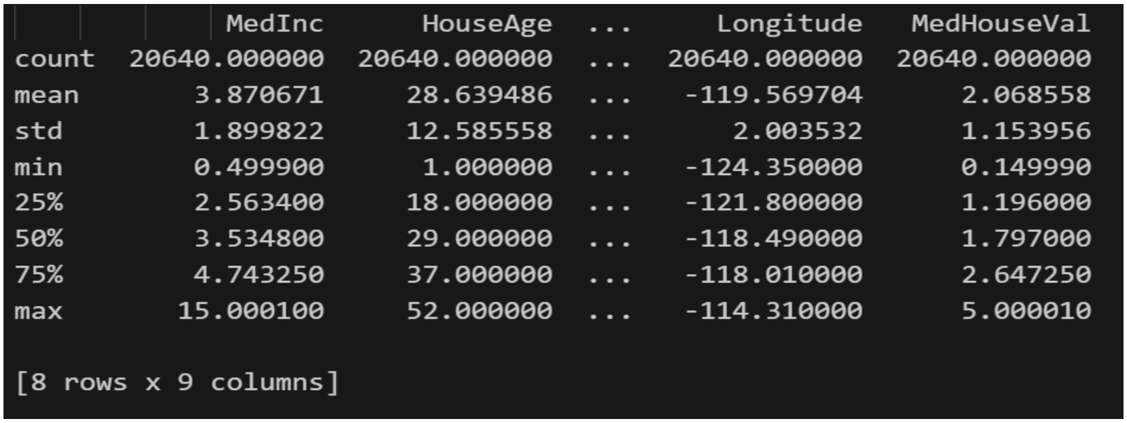
➢ Other Outputs:



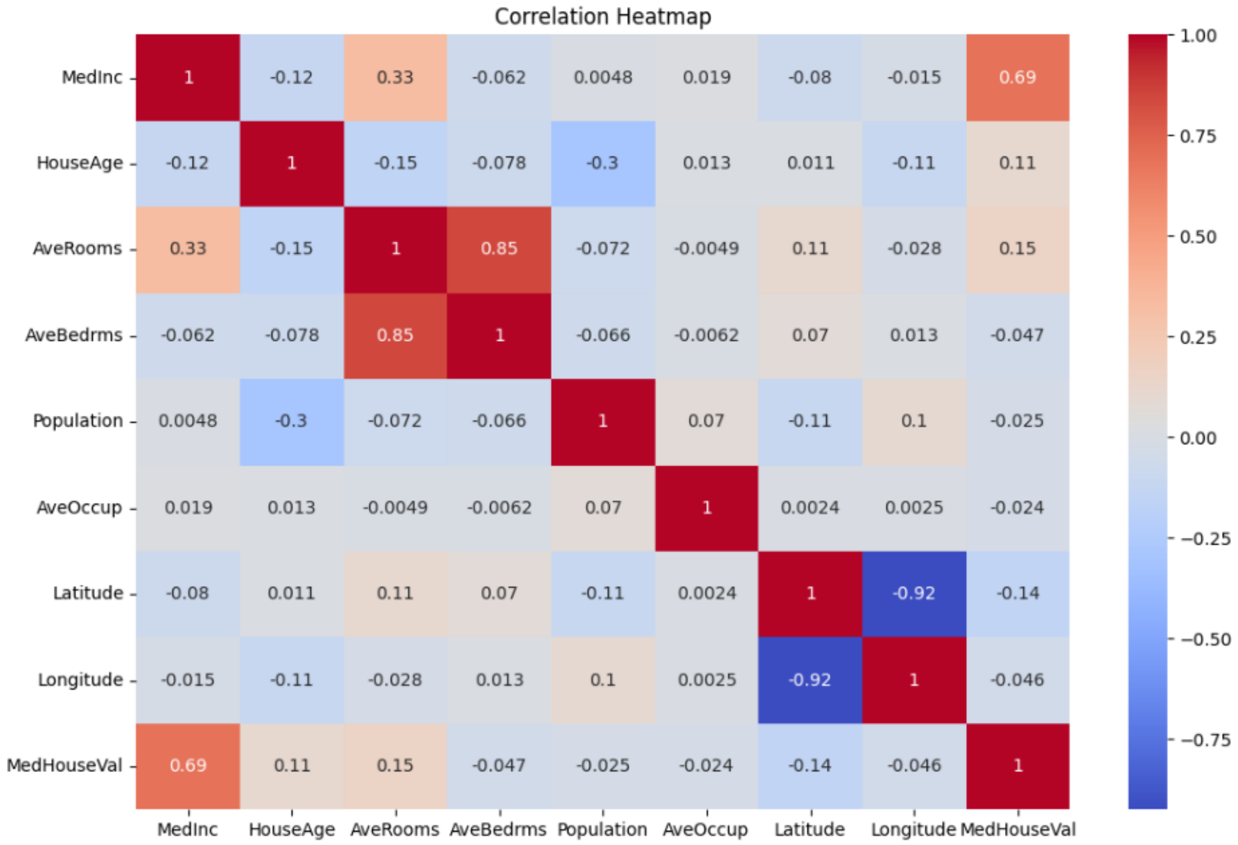
```
   MedInc  HouseAge  AveRooms  ...  Latitude  Longitude  MedHouseVal
0  8.3252      41.0  6.984127  ...     37.88    -122.23        4.526
1  8.3014      21.0  6.238137  ...     37.86    -122.22        3.585
2  7.2574      52.0  8.288136  ...     37.85    -122.24        3.521
3  5.6431      52.0  5.817352  ...     37.85    -122.25        3.413
4  3.8462      52.0  6.281853  ...     37.85    -122.25        3.422
```



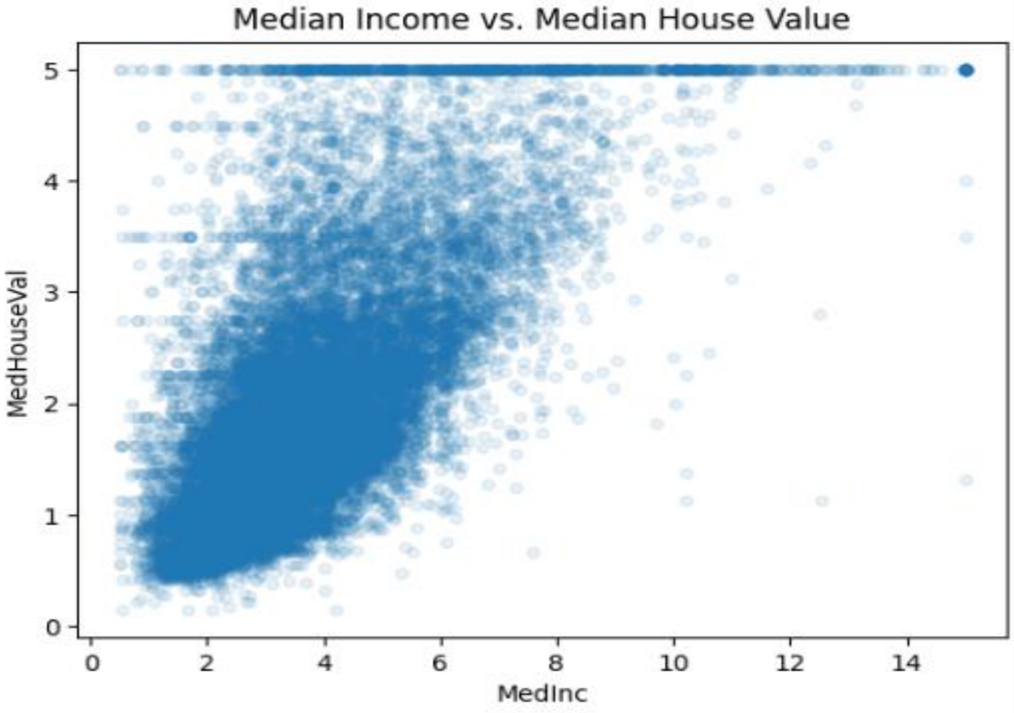
```
[5 rows x 9 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

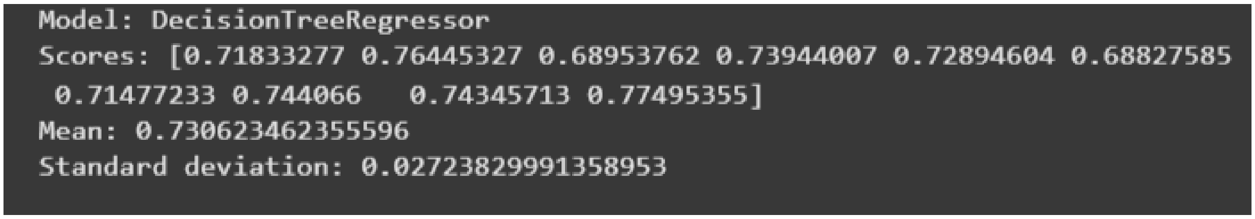


```
             MedInc      HouseAge  ...     Longitude   MedHouseVal
count  20640.000000  20640.000000  ...  20640.000000  20640.000000
mean       3.870671     28.639486  ...   -119.569704      2.068558
std        1.899822     12.585558  ...      2.003532      1.153956
min        0.499900      1.000000  ...   -124.350000      0.149990
25%        2.563400     18.000000  ...   -121.800000      1.196000
50%        3.534800     29.000000  ...   -118.490000      1.797000
75%        4.743250     37.000000  ...   -118.010000      2.647250
max       15.000100     52.000000  ...   -114.310000      5.000010

[8 rows x 9 columns]
```

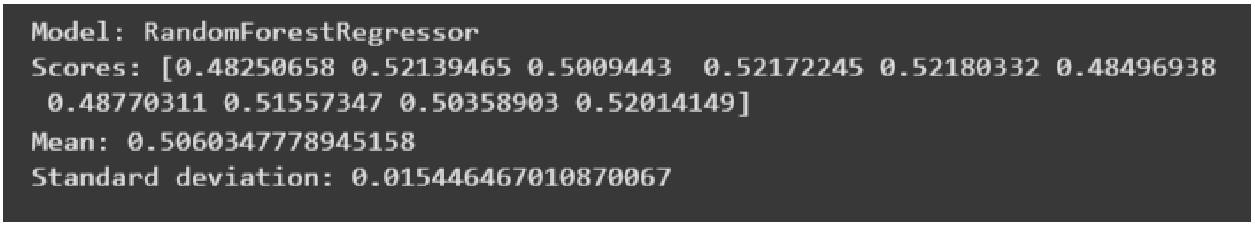


	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
MedInc	1	-0.12	0.33	-0.062	0.0048	0.019	-0.08	-0.015	0.69
HouseAge	-0.12	1	-0.15	-0.078	-0.3	0.013	0.011	-0.11	0.11
AveRooms	0.33	-0.15	1	0.85	-0.072	-0.0049	0.11	-0.028	0.15
AveBedrms	-0.062	-0.078	0.85	1	-0.066	-0.0062	0.07	0.013	-0.047
Population	0.0048	-0.3	-0.072	-0.066	1	0.07	-0.11	0.1	-0.025
AveOccup	0.019	0.013	-0.0049	-0.0062	0.07	1	0.0024	0.0025	-0.024
Latitude	-0.08	0.011	0.11	0.07	-0.11	0.0024	1	-0.92	-0.14
Longitude	-0.015	-0.11	-0.028	0.013	0.1	0.0025	-0.92	1	-0.046
MedHouseVal	0.69	0.11	0.15	-0.047	-0.025	-0.024	-0.14	-0.046	1

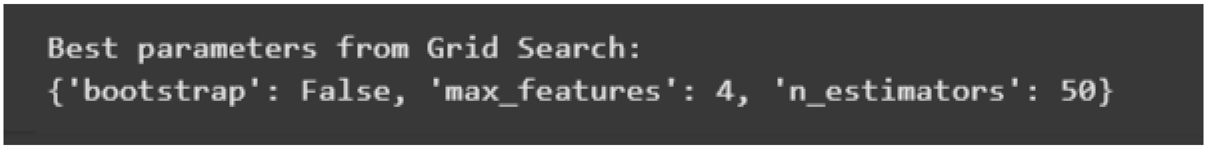




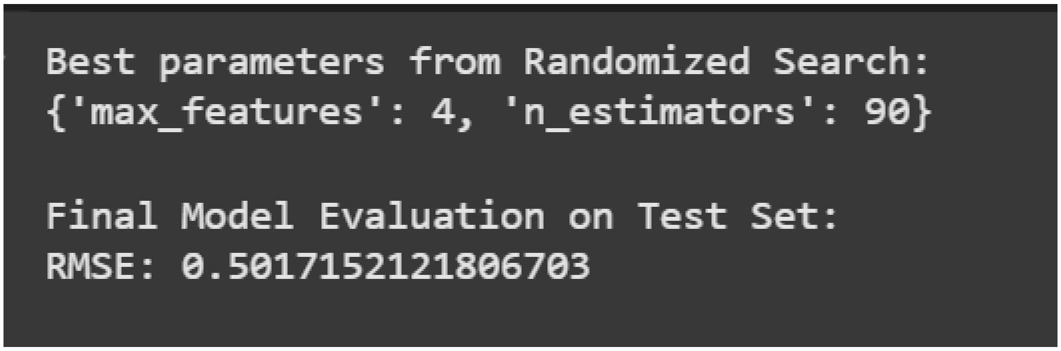
```
Model: DecisionTreeRegressor
Scores: [0.71833277 0.76445327 0.68953762 0.73944007 0.72894604 0.68827585
 0.71477233 0.744066   0.74345713 0.77495355]
Mean: 0.730623462355596
Standard deviation: 0.02723829991358953
```



```
Model: RandomForestRegressor
Scores: [0.48250658 0.52139465 0.5009443  0.52172245 0.52180332 0.48496938
 0.48770311 0.51557347 0.50358903 0.52014149]
Mean: 0.5060347778945158
Standard deviation: 0.015446467010870067
```



```
Best parameters from Grid Search:
{'bootstrap': False, 'max_features': 4, 'n_estimators': 50}
```



```
Best parameters from Randomized Search:
{'max_features': 4, 'n_estimators': 90}

Final Model Evaluation on Test Set:
RMSE: 0.5017152121806703
```