ARCH TECHNOLOGIES

AI/ML INTERNSHIP

Submitted By:
Ume Taqadus
Category B
Manual 1
Intern ID: ARCH-2506-0327
GitHub: https://github.com/Taqadus842

CHAPTER 1

EXERCISE

1. How would you define Machine Learning?

Machine Learning is a field of artificial intelligence where systems learn from data to make predictions or decisions without being explicitly programmed.

2. Can you name four types of problems where it shines?

- Spam detection
- Product recommendation
- Image recognition
- Fraud detection

3. What is a labeled training set?

A labeled training set is a dataset where each example includes both the input features and the correct output (label), used for supervised learning.

4. What are the two most common supervised tasks?

- Classification (e.g., spam vs. not spam)
- Regression (e.g., predicting house prices)

5. Can you name four common unsupervised tasks?

- Clustering (e.g., customer segmentation)
- Dimensionality reduction (e.g., PCA)
- Anomaly detection
- Association rule learning (e.g., market basket analysis)

6. What type of Machine Learning algorithm would you use to allow a robot to walk in various unknown terrains?

Reinforcement Learning, as the robot learns by trial and error in a dynamic environment.

7. What type of algorithm would you use to segment your customers into multiple groups?

An **unsupervised clustering algorithm**, such as **K-Means** or **DBSCAN**.

8. Would you frame the problem of spam detection as a supervised or an unsupervised learning problem?

Supervised learning, because we have labeled examples of spam and non-spam emails.

9. What is an online learning system?

An online learning system learns incrementally from a continuous stream of data, adapting quickly to changes in real time.

10. What is out-of-core learning?

Out-of-core learning handles data that is too large to fit in memory by using algorithms that train on mini-batches from the disk.

11. What type of learning algorithm relies on a similarity measure to make predictions?

Instance-based learning algorithms, like **k-Nearest Neighbors (k-NN)**.

12. What is the difference between a model parameter and a learning algorithm's hyperparameter?

- **Model parameter**: Learned from data (e.g., weights in linear regression).
- **Hyperparameter**: Set before training (e.g., learning rate, number of trees in a random forest).

13. What do model-based learning algorithms search for? What is the most common strategy they use to succeed? How do they make predictions?

They search for the **best parameters** to fit the model to the training data.
The most common strategy is to **minimize a cost function** (e.g., MSE).
They make predictions using the learned model on new data.

14. Can you name four of the main challenges in Machine Learning?

- Overfitting
- Underfitting
- Data quality and quantity
- Non-representative training data

15. If your model performs great on the training data but generalizes poorly to new instances, what is happening? Can you name three possible solutions?

This is **overfitting**.
Possible solutions:

- Use simpler model (reduce complexity)
- Add regularization
- Get more training data

16. What is a test set and why would you want to use it?

A test set is unseen data used to evaluate the final model's performance. It checks how well the model generalizes.

17. What is the purpose of a validation set?

To tune hyperparameters and select the best model without touching the test set.

18. What can go wrong if you tune hyperparameters using the test set?

The model may overfit to the test set, making evaluation results misleading and not reflective of real-world performance.

19. What is repeated cross-validation and why would you prefer it to using a single validation set?

Repeated cross-validation splits the data into multiple train/validation sets to ensure a more **reliable and stable evaluation**, reducing variance caused by a single split.

CHAPTER 2

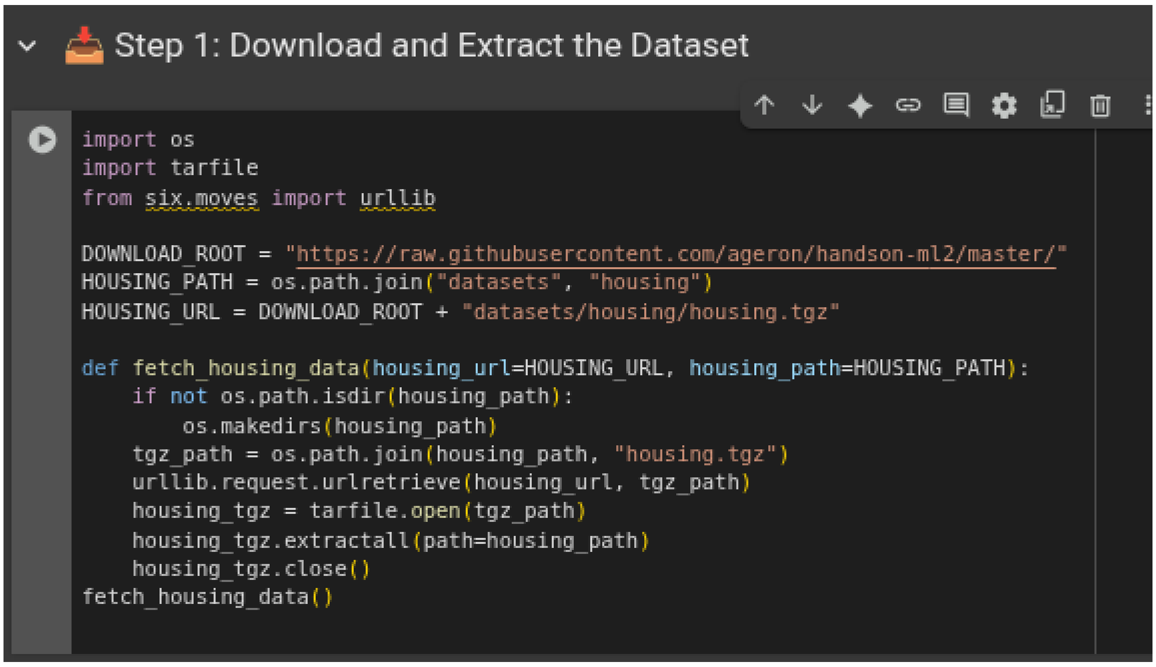
Code

🔹 Step 1: Import Required Libraries

We import libraries like NumPy, Pandas for data handling, and Matplotlib, Seaborn for data visualization.

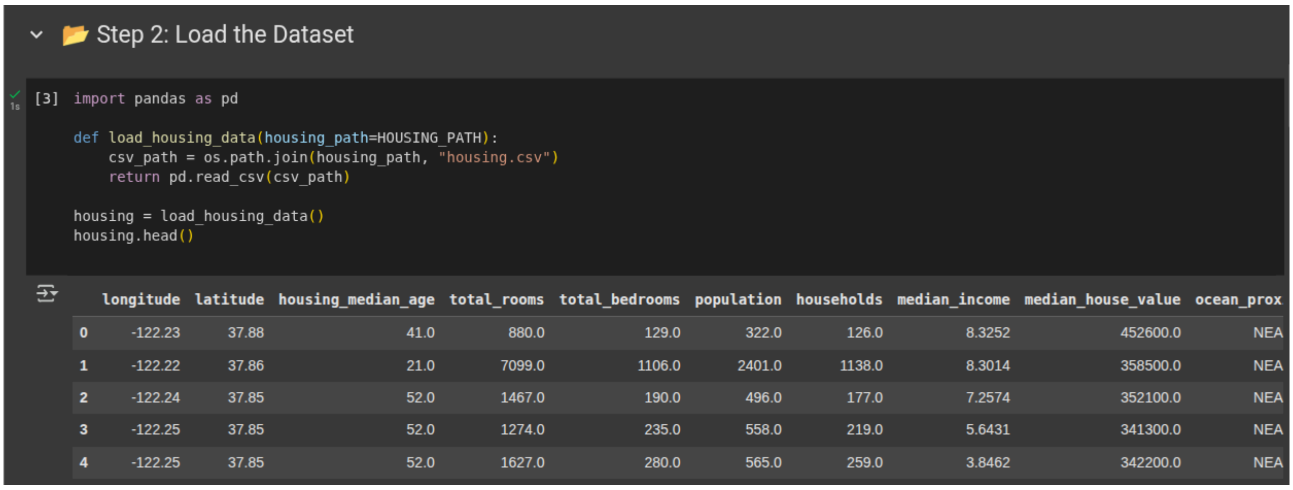
🔹 Step 2: Load Dataset

We load the California Housing dataset using `fetch_california_housing` and convert it into a DataFrame for easier analysis.



Step 1: Download and Extract the Dataset

```python
import os
import tarfile
from six.moves import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
fetch_housing_data()
```



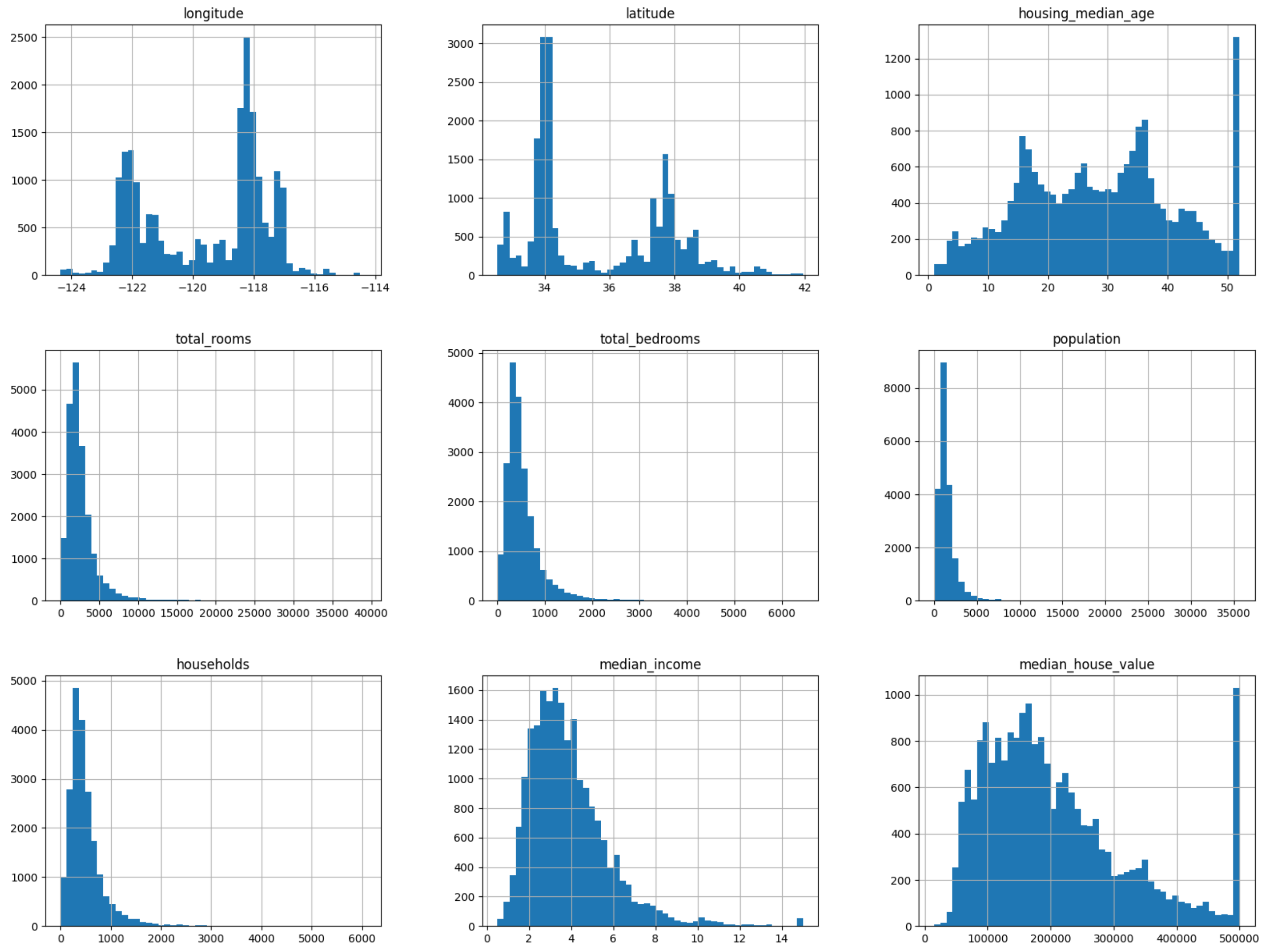
Step 2: Load the Dataset

```python
[3] import pandas as pd

    def load_housing_data(housing_path=HOUSING_PATH):
        csv_path = os.path.join(housing_path, "housing.csv")
        return pd.read_csv(csv_path)

    housing = load_housing_data()
    housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_prox
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEA
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEA
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEA
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEA
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEA

🔹 Step 3: Visualize Dataset

We plot histograms of each column to understand the distribution of features and check if normalization or preprocessing is needed.

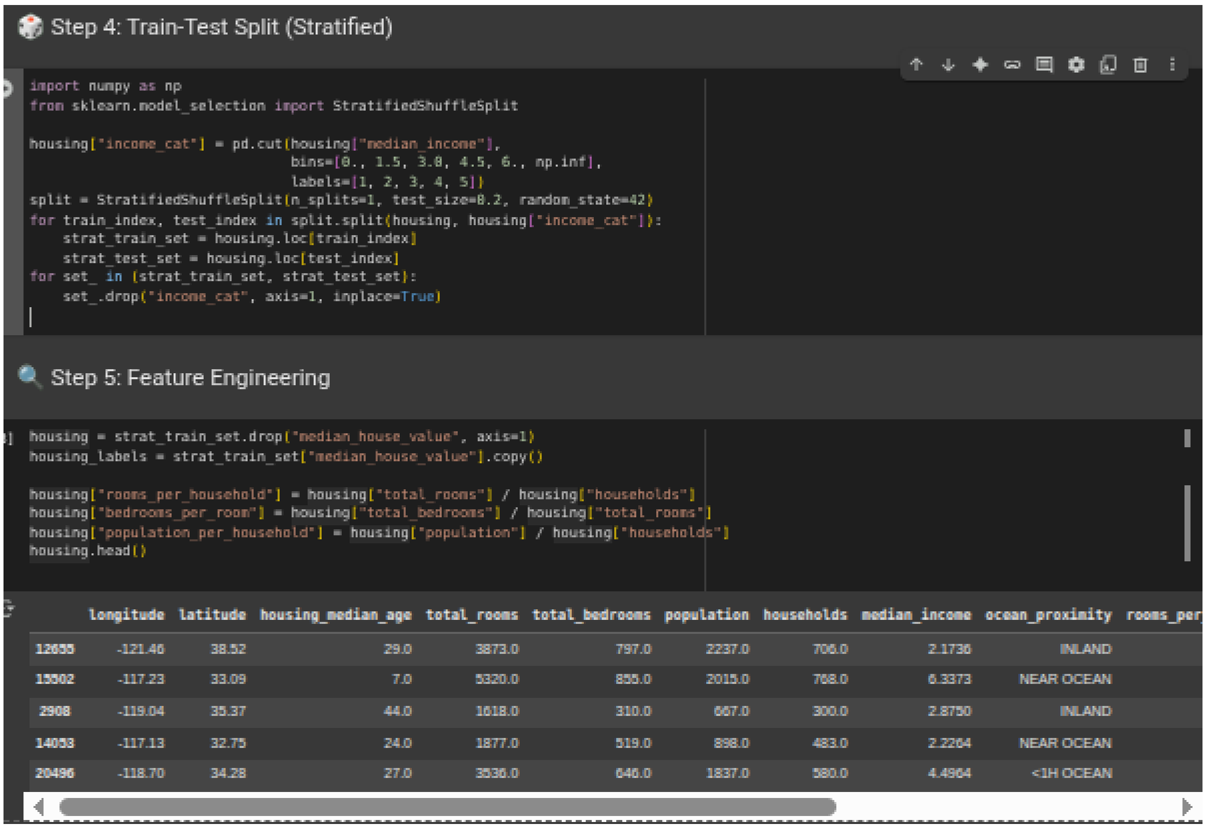


🔹 Step 4: Train-Test Split Using Stratified Sampling

We create a new column `income_cat` for stratified sampling based on `MedInc` to ensure the income distribution is preserved in both training and testing datasets.

🔹 Step 5: Separate Labels and Features

We remove the target column `MedHouseVal` from the dataset and store it in a separate variable as labels.



Step 4: Train-Test Split (Stratified)

```
import numpy as np
from sklearn.model_selection import StratifiedShuffleSplit

housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

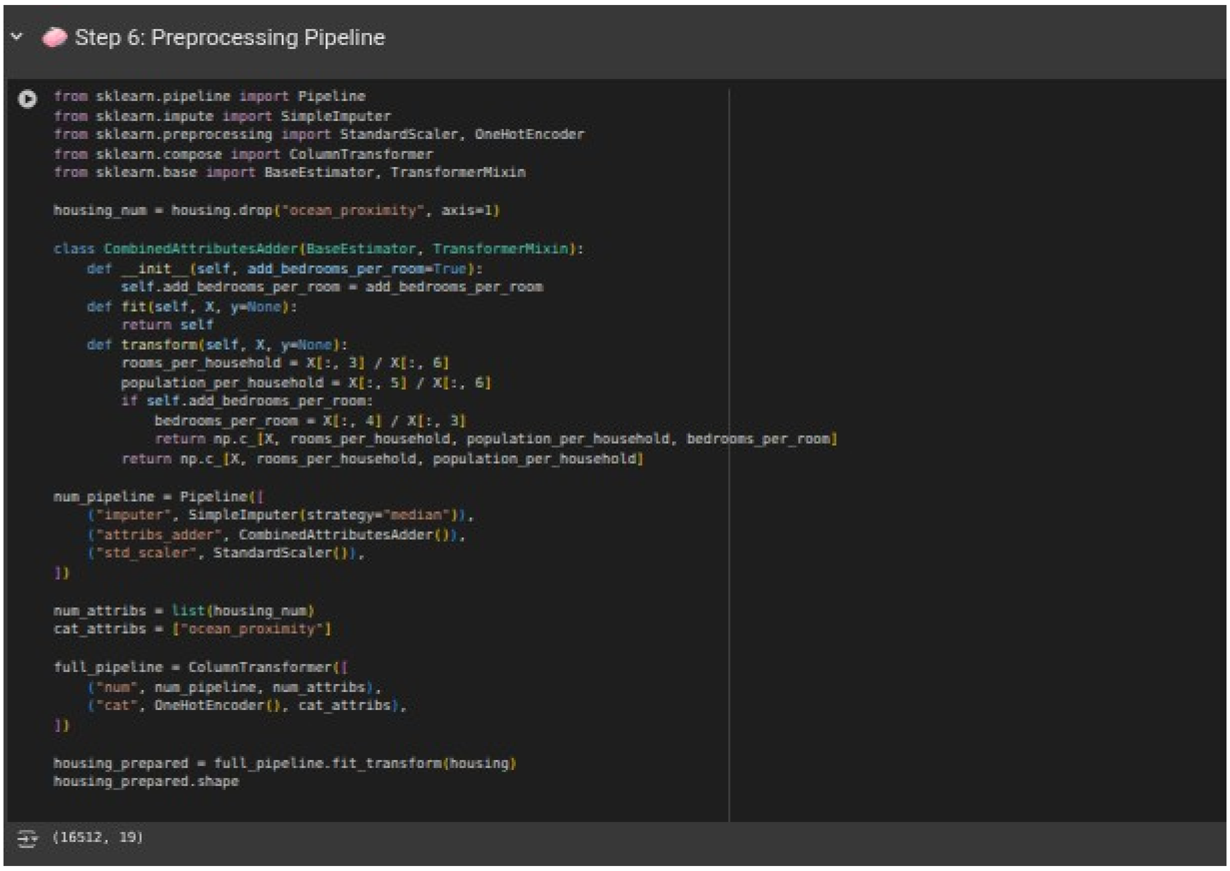
Step 5: Feature Engineering

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()

housing["rooms_per_household"] = housing["total_rooms"] / housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"] / housing["total_rooms"]
housing["population_per_household"] = housing["population"] / housing["households"]
housing.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	rooms_per
12655	-121.46	38.52	29.0	3873.0	797.0	2237.0	706.0	2.1736	INLAND	
15502	-117.23	33.09	7.0	5320.0	855.0	2015.0	768.0	6.3373	NEAR OCEAN	
2908	-119.04	35.37	44.0	1618.0	310.0	667.0	300.0	2.8750	INLAND	
14053	-117.13	32.75	24.0	1877.0	519.0	898.0	483.0	2.2264	NEAR OCEAN	
20496	-118.70	34.28	27.0	3536.0	646.0	1837.0	580.0	4.4964	<1H OCEAN	

🔹 Step 6: Create Preprocessing Pipeline

We create a pipeline that first fills missing values with the median and then standardizes the numerical values using `StandardScaler`.



Step 6: Preprocessing Pipeline

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, TransformerMixin

housing_num = housing.drop("ocean_proximity", axis=1)

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room=True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        rooms_per_household = X[:, 3] / X[:, 6]
        population_per_household = X[:, 5] / X[:, 6]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, 4] / X[:, 3]
            return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]
        return np.c_[X, rooms_per_household, population_per_household]

num_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("attribs_adder", CombinedAttributesAdder()),
    ("std_scaler", StandardScaler()),
])

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OneHotEncoder(), cat_attribs),
])

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

```
(16512, 19)
```

🔹 Step 7: Train Linear Regression Model

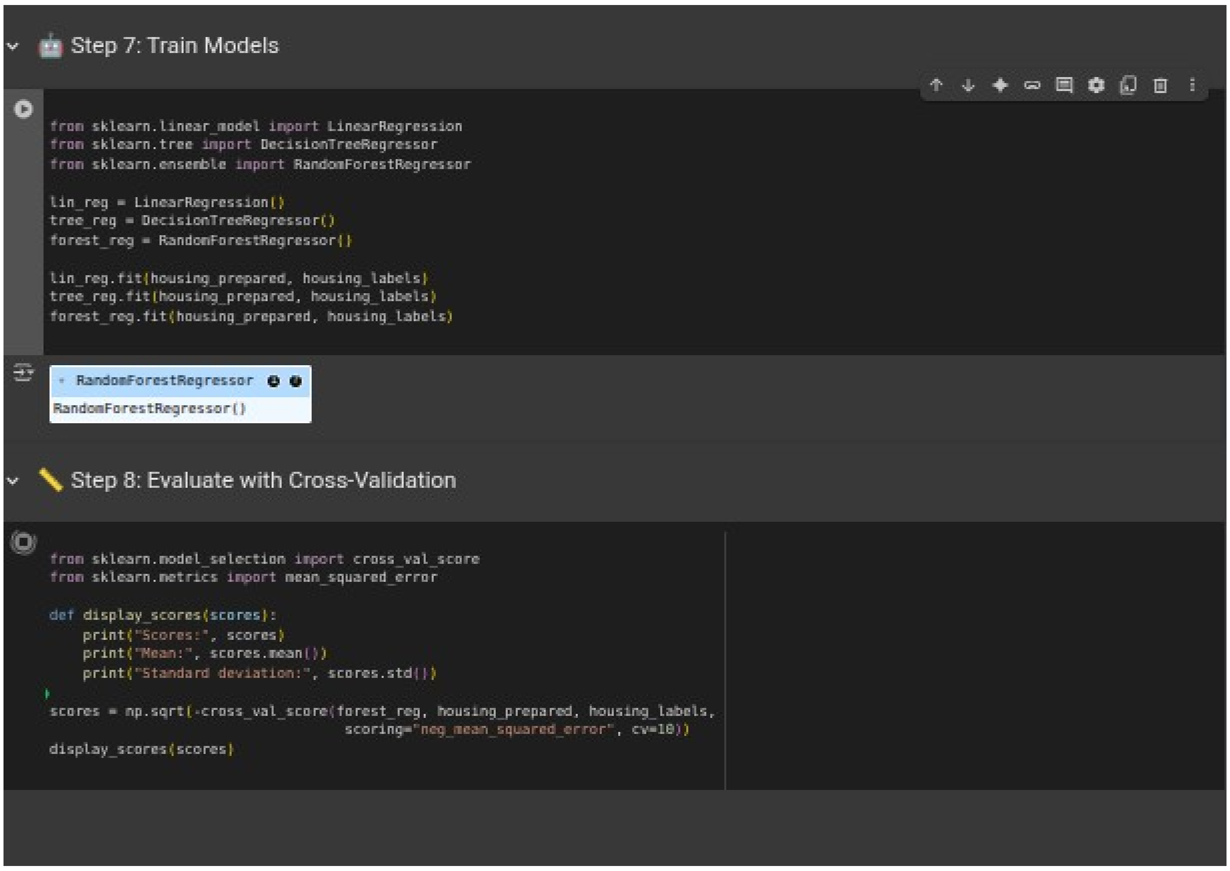
We use the preprocessed training data to train a Linear Regression model using `fit()` method.

🔹 Step 8: Evaluate Linear Regression Model

We calculate RMSE (Root Mean Squared Error) by comparing model predictions and actual values using `mean_squared_error`.

🔹 Step 9: Cross Validation

We perform 10-fold cross-validation to validate the model performance more reliably by calculating negative MSE and then converting it to RMSE.



🤖 Step 7: Train Models

```python
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

lin_reg = LinearRegression()
tree_reg = DecisionTreeRegressor()
forest_reg = RandomForestRegressor()

lin_reg.fit(housing_prepared, housing_labels)
tree_reg.fit(housing_prepared, housing_labels)
forest_reg.fit(housing_prepared, housing_labels)
```

```
RandomForestRegressor
RandomForestRegressor()
```

📏 Step 8: Evaluate with Cross-Validation

```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error

def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

scores = np.sqrt(-cross_val_score(forest_reg, housing_prepared, housing_labels,
                                  scoring="neg_mean_squared_error", cv=10))
display_scores(scores)
```

🔹 Step 10: Grid Search for Best Parameters

We use `GridSearchCV` with `RandomForestRegressor` to try different combinations of `n_estimators` and `max_features` to find the best model.

🔹 Step 11: Feature Importance

We get the best model from grid search and extract the feature importances to know which features influence the prediction most.

🔹 Step 12: Evaluate on Test Data

We use the final model to make predictions on the test set and calculate the final RMSE to check how well our model performs on unseen data.

🔹 Step 13: Confidence Interval

We compute a 95% confidence interval to estimate the range in which our prediction error lies.

🔹 Step 14: Save Final Model

We save the best trained model using `joblib` so we can use it later without retraining.
