Task 1

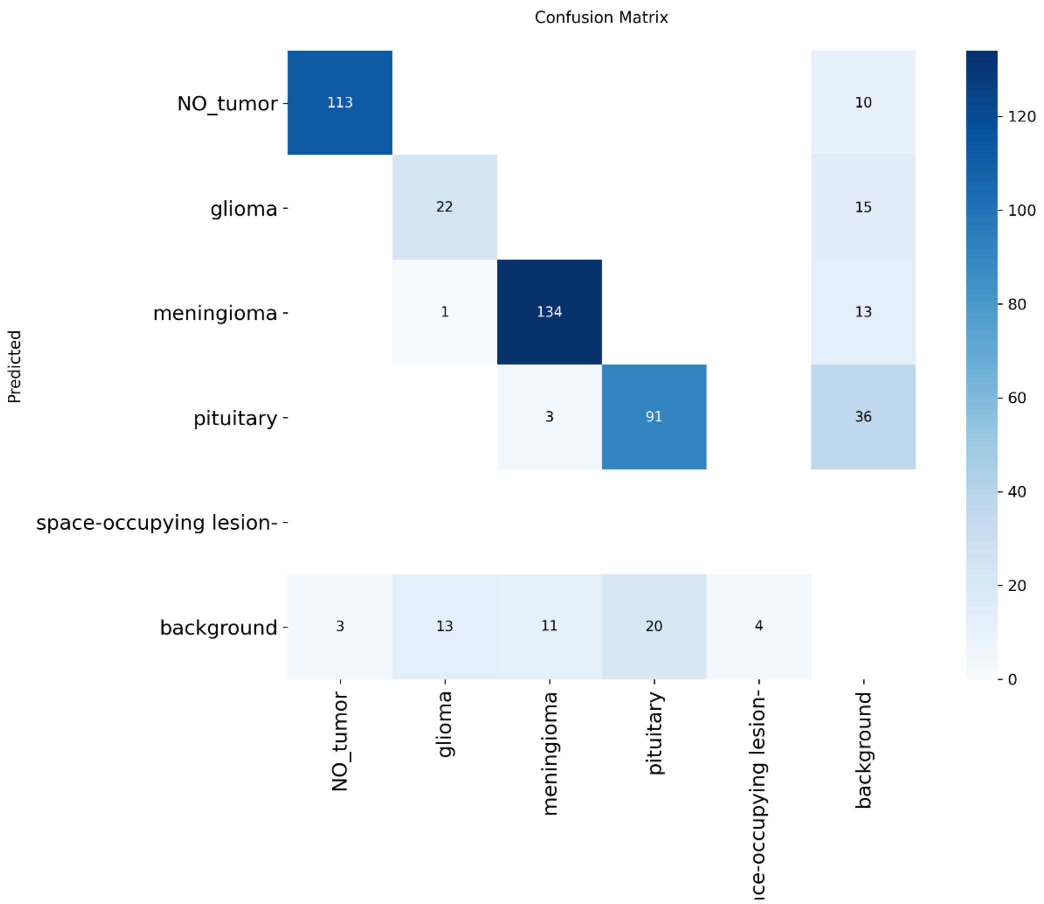
Project- Brain Tumor Segmentation with YOLO 11 and SAM

```
from google.colab import files
uploaded = files.upload()
import zipfile
import os

with zipfile.ZipFile("Tumor Detection.zip", 'r') as zip_ref:
    zip_ref.extractall("Tumor Detection")
!pip install ultralytics
from ultralytics import YOLO

# Load a model
model = YOLO("yolo11n.pt")

# Train the model
train_results = model.train(
    data="/content/Tumor Detection/data.yaml",  # path to dataset YAML
    epochs=20,  # number of training epochs
    imgsz=640,  # training image size
    device=0,  # device to run on, i.e. device=0 or device=0,1,2,3 or device=cpu, train model
on gpu not on cpu
)
from google.colab import files
uploaded = files.upload()
import zipfile
import os
```

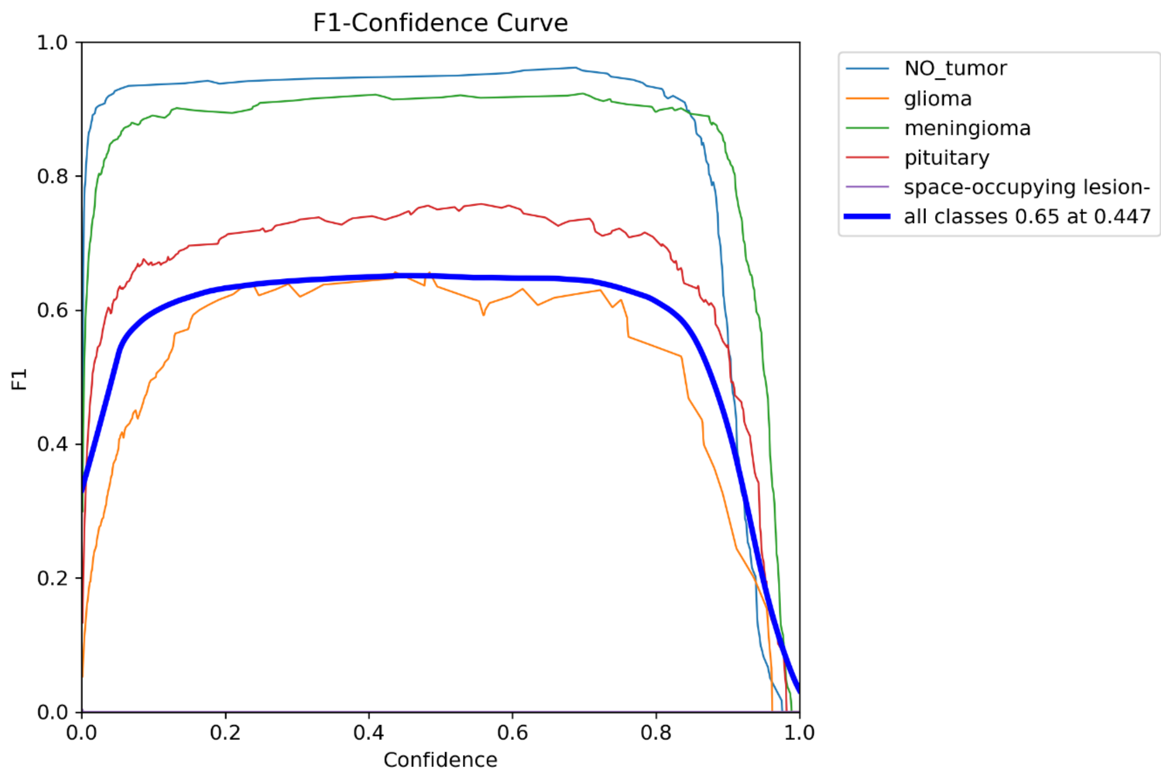
```
with zipfile.ZipFile("test_images.zip", 'r') as zip_ref:
    zip_ref.extractall("test_images")
from ultralytics import YOLO

# Load a model
model = YOLO("/content/runs/detect/train/weights/best.pt")
# Output of YOLO11 (Bounding Box) now becomes the input of SAM2 Model for
INTANCE SEGMENTATION
# Perform object detection on an image
results = model("/content/test_images/test_images/meningioma_3.jpg", save=True)
results[0].show()
from ultralytics import YOLO

# Load a model
model = YOLO("/content/runs/detect/train/weights/best.pt")
# Output of YOLO11 (Bounding Box) now becomes the input of SAM2 Model for
INTANCE SEGMENTATION
# Perform object detection on an image
results = model("/content/test_images/test_images", save=True)
from ultralytics import YOLO
# For Box Coordinates
# Load a model
model = YOLO("runs/detect/train/weights/best.pt")  # pretrained YOLO11n model

# Run batched inference on a list of images
results = model("test_images/test_images/glioma_2.jpg")  # return a list of Results objects

# Process results list
for result in results:
```

```
    boxes = result.boxes  # Boxes object for bounding box outputs
    print(boxes)
from ultralytics import YOLO
from ultralytics import SAM

# Load the YOLO model
yolo_model = YOLO("runs/detect/train/weights/best.pt")  # pretrained YOLO model

# Run batched inference on a list of images
results = yolo_model("/content/test_images/test_images/meningioma_3.jpg")  # return a list of Results objects

# Load the SAM model for intance segmentation
sam_model = SAM("sam2_b.pt") # SAM Model is of Meta but it also integrate into ultalytics package, so need no to install it seperately
#sam2_b "b for base model, one of the category of sam 2 model"
for result in results:
    class_ids = result.boxes.cls.int().tolist()  # noqa
    if len(class_ids):
        boxes = result.boxes.xyxy  # Boxes object for bbox outputs
        sam_results = sam_model(result.orig_img, bboxes=boxes, verbose=False, save=True, device=0)
```

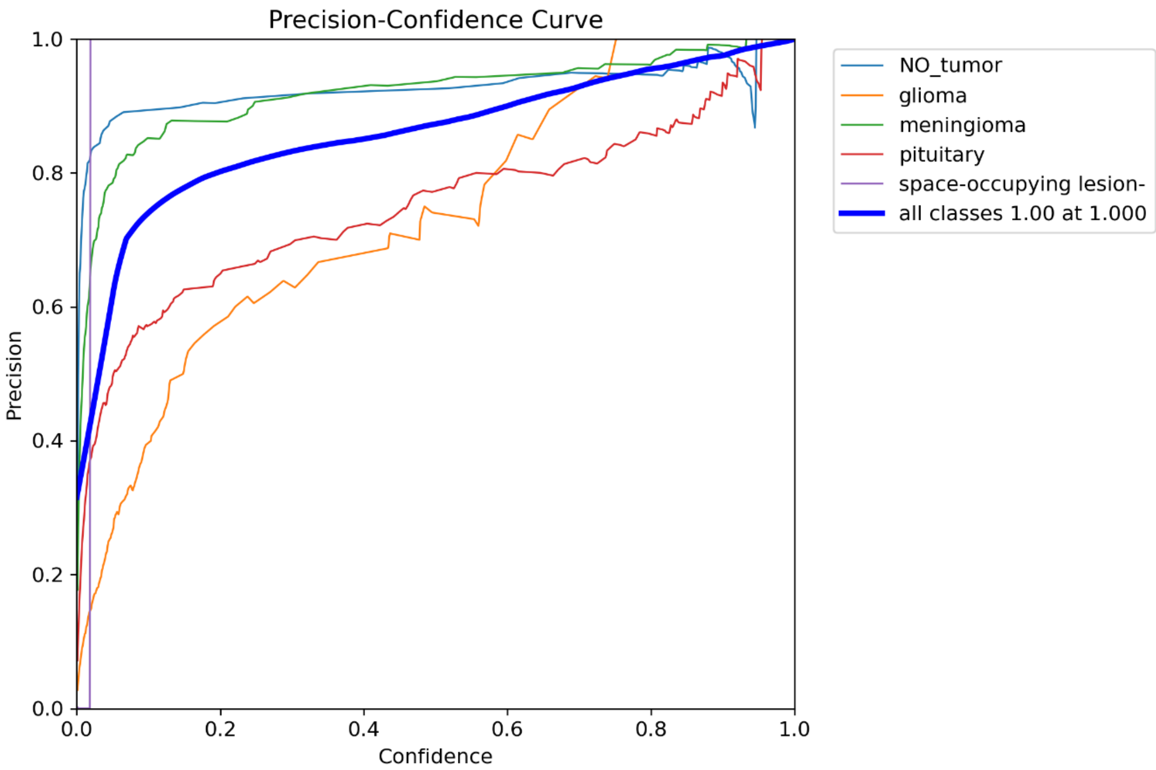
Screenshots or Results (Brain Tumor Detection)

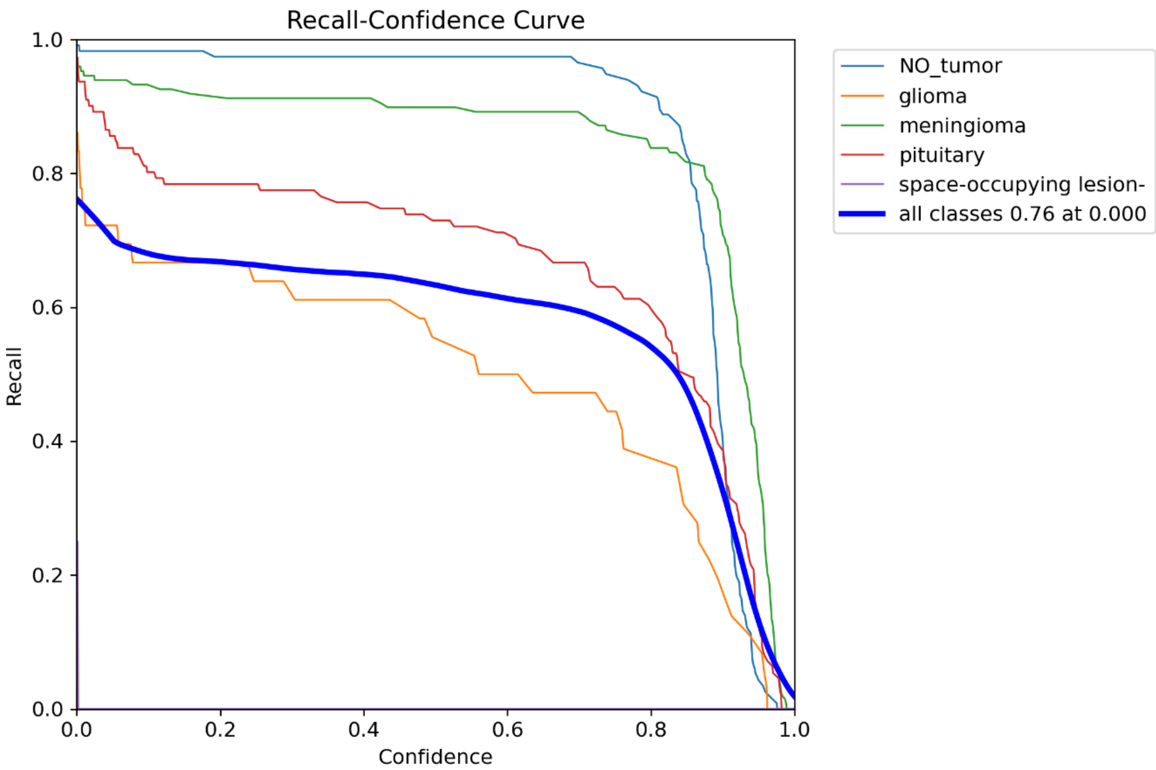


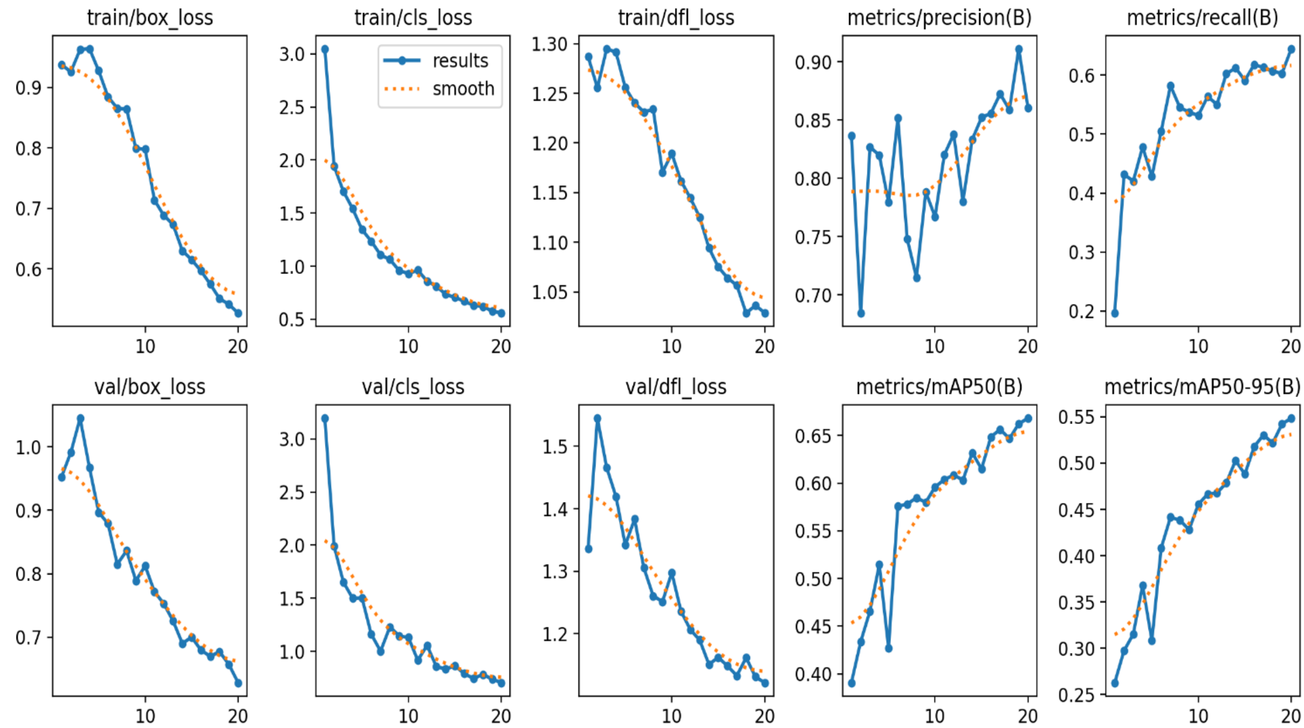
Confusion Matrix
Predicted
NO_tumor
glioma
meningioma
pituitary
space-occupying lesion-
background
113
10
22
15
1
134
13
3
91
36
3
13
11
20
4
NO_tumor
glioma
meningioma
pituitary
ice-occupying lesion-
background
120
100
80
60
40
20
0

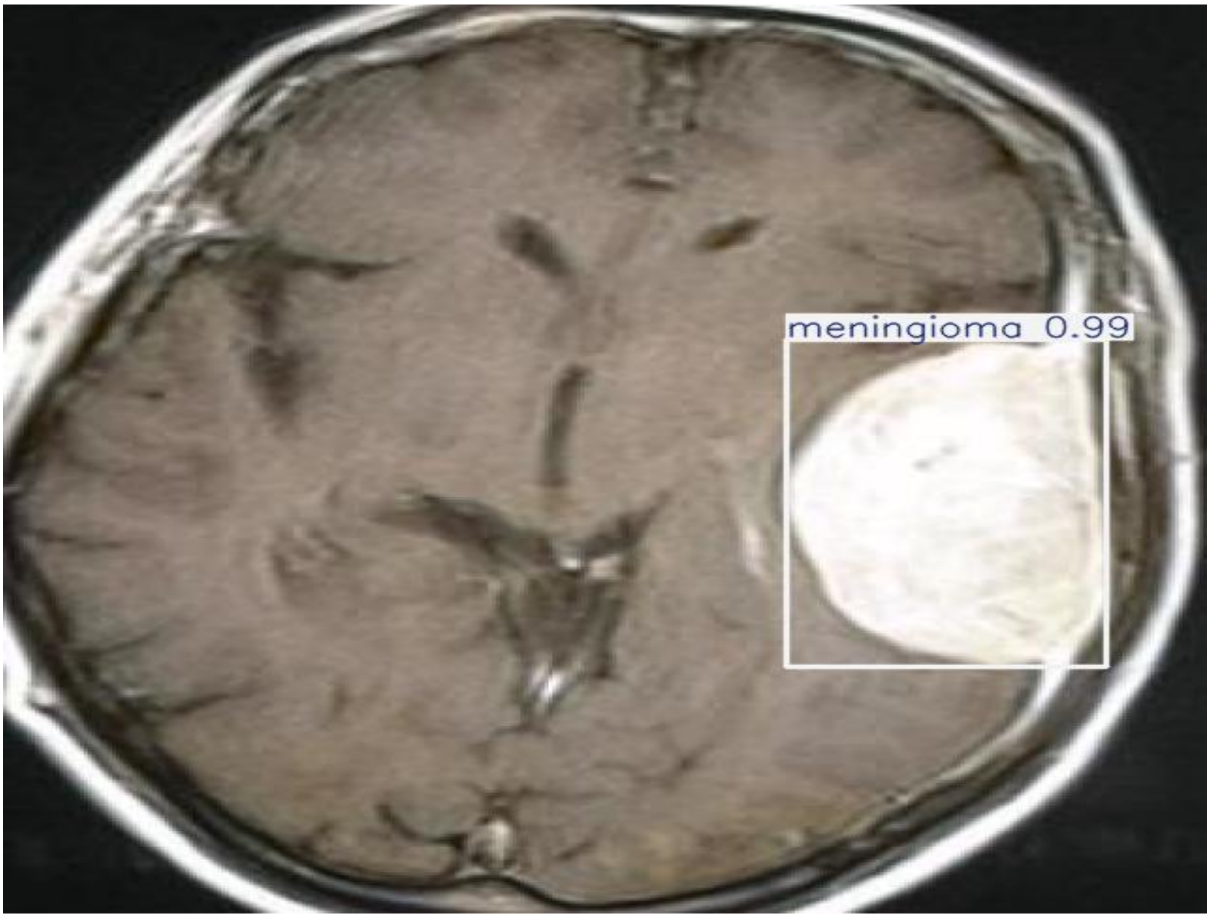


F1-Confidence Curve
F1
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
NO_tumor
glioma
meningioma
pituitary
space-occupying lesion-
all classes 0.65 at 0.447


Precision-Confidence Curve
Precision
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
NO_tumor
glioma
meningioma
pituitary
space-occupying lesion-
all classes 1.00 at 1.000


Recall-Confidence Curve
Recall
Confidence
0.0
0.2
0.4
0.6
0.8
1.0
NO_tumor
glioma
meningioma
pituitary
space-occupying lesion-
all classes 0.76 at 0.000


train/box_loss
train/cls_loss
results
smooth
train/dfl_loss
metrics/precision(B)
metrics/recall(B)
val/box_loss
val/cls_loss
val/dfl_loss
metrics/mAP50(B)
metrics/mAP50-95(B)


meningioma 0.99

Machine Learning for Everybody– Full Course by freecodecamp.org

K-Nearest Neighbors (KNN)– Concept and implementation

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

from sklearn.preprocessing import StandardScaler

from imblearn.over_sampling import RandomOverSampler

cols = ["fLength", "fWidth", "fSize", "fConc", "fConc1", "fAsym", "fM3Long", "fM3Trans", "fAlpha", "fDist", "class"]

df = pd.read_csv("magic04.data", names=cols) # Assign labels to columns of this dataset

df.head()

df["class"].unique() # g for gamma and h for hardons

# convert class data type object to numbers

df["class"] = (df["class"] == 'g').astype(int)
```

10 features are used to tain the model for predicting target variable

---

[13]

1s

```
for label in cols[:-1]:

  plt.hist(df[df["class"]==1][label], color='blue', label='gamma', alpha=0.7, density=True)

  plt.hist(df[df["class"]==0][label], color='red', label='hadron', alpha=0.7, density=True)

  plt.title(label)

  plt.ylabel("Prabability")

  plt.xlabel(label)

  plt.legend()
```

plt.show()

Train, Validation, Test Datasets

shuffle the data

split 60% for training data, 20% (everything b/w 60 & 80%) for validation data

20% (everything b/w 80 & 100%) for testing data

train, valid, test = np.split(df.sample(frac=1), [int(0.6*len(df)), int(0.8*len(df))])

Scale the data, some values like flenth, fwidth are lare while other are so smale in 4. some numbers or even 0. sometime which affects the result so scaled it

```
def scale_dataset(dataframe, oversample=False):
  # X contains all the feature columns (except the last column, which is the target)
  X = dataframe[dataframe.columns[:-1]].values
  # y contains the target column (the last column)
  y = dataframe[dataframe.columns[-1]].values
  scaler = StandardScaler()
  X = scaler.fit_transform(X)

  # Take less class and keep samplin from there to increase the size of our dataset of that smaller class so that they now match
  if oversample:
    ros = RandomOverSampler()
    X, y = ros.fit_resample(X, y)

  # FOR combine X with y, makes y 2D as of X, by reshpe(-1,1), the last one is used to make it column vector
  data = np.hstack((X, np.reshape(y, (-1, 1))))
  return data, X, y
```

Unequal Datset (gamma has more rows(data), as compared to hadron (outliers))

```
print(len(train[train["class"]==1])) # gamma
print(len(train[train["class"]==0])) # hadron
```

train, X_train, y_train = scale_dataset(train, oversample=True)

sum(y_train == 1)

sum(y_train == 0)

Now both gamma and hadron have same number of values

Donot oversample or make same number of values to gamma and hadron in valid and test dataset beacause it's unseen data and used for checking model accuracy

valid, X_valid, y_valid = scale_dataset(valid, oversample=False)

test, X_test, y_test = scale_dataset(test, oversample=False)

K-Nearest Neighbour ML Classification Algorithm

from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=3)

knn_model.fit(X_train, y_train)

y_pred = knn_model.predict(X_test)

y_pred

from sklearn.metrics import classification_report

print(classification_report(y_test,y_pred))

```
precision    recall  f1-score   support

           0       0.75      0.72      0.74      1358
           1       0.85      0.87      0.86      2446

    accuracy                           0.81      3804
   macro avg       0.80      0.79      0.80      3804
weighted avg       0.81      0.81      0.81      3804
```

• Naive Bayes– Theory and application in classification tasks

from sklearn.naive_bayes import GaussianNB

nb_model = GaussianNB()

nb_model = nb_model.fit(X_train, y_train)

y_pred = nb_model.predict(X_test)

print(classification_report(y_test,y_pred)) # Model is not good as compared to K-Nearest Neighbor Algorithm w.r.t to this dataset

• Logistic Regression– Understanding and implementation.

from sklearn.linear_model import LogisticRegression

```
lg_model = LogisticRegression()

lg_model = lg_model.fit(X_train, y_train)

y_pred = lg_model.predict(X_test)

print(classification_report(y_test,y_pred))  # Model is better than Naive Bayes but not good as KNN w.r.t this dataset
```

```
precision    recall  f1-score   support

           0       0.68      0.72      0.70      1358
           1       0.84      0.81      0.83      2446

    accuracy                           0.78      3804
   macro avg       0.76      0.77      0.76      3804
weighted avg       0.78      0.78      0.78      3804
```

• **Support Vector Machines (SVM)– Introduction and practical use**

```
from sklearn.svm import SVC

svm_model = SVC()

svm_model = svm_model.fit(X_train, y_train)

y_pred = svm_model.predict(X_test)

print(classification_report(y_test,y_pred))  # Model is best than other previous algoirthm w.r.t this dataset, nott good with outliers in the dataset
```

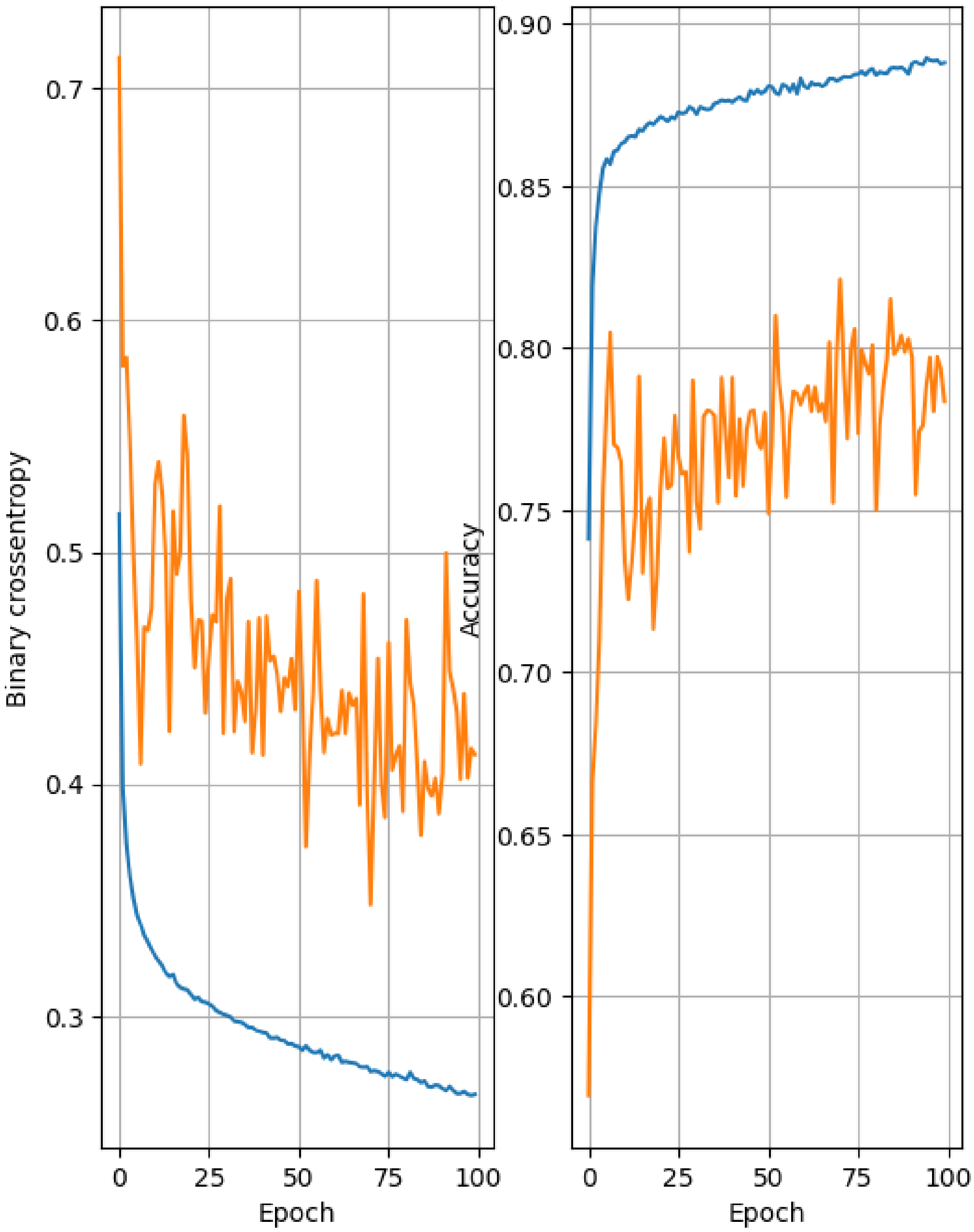
```
precision    recall  f1-score   support

           0       0.81      0.79      0.80      1358
           1       0.89      0.90      0.89      2446

    accuracy                           0.86      3804
   macro avg       0.85      0.85      0.85      3804
weighted avg       0.86      0.86      0.86      3804
```

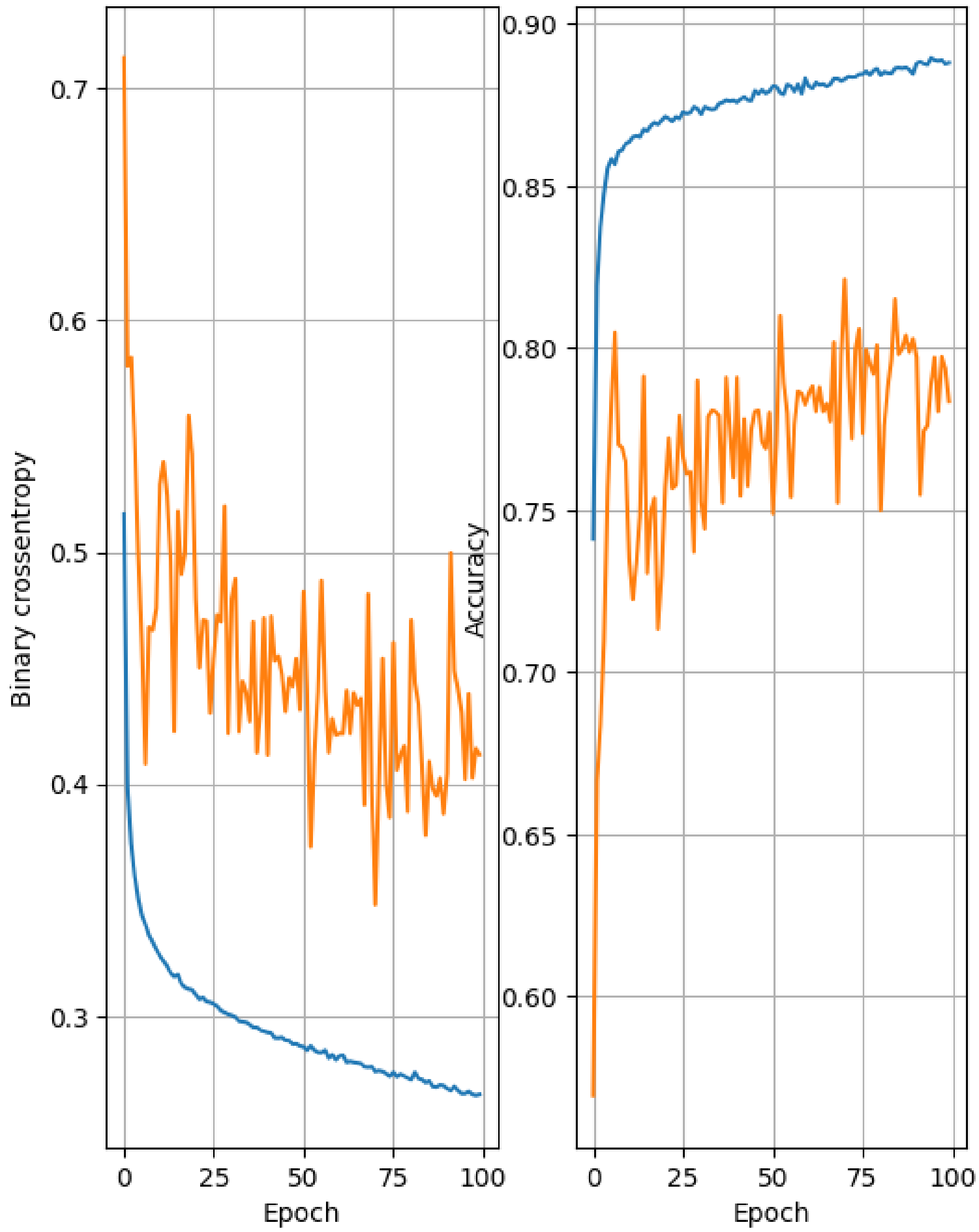
• **Neural Networks– Basics, TensorFlow usage, and model training**

```
import tensorflow as tf

import matplotlib.pyplot as plt

def plot_history(history):

  # plot on my axis

  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 8))

  ax1.plot(history.history['loss'], label='loss')

  ax1.plot(history.history['val_loss'], label='val_loss')
```

```
  ax1.set_xlabel('Epoch')
  ax1.set_ylabel('Binary crossentropy')
  ax1.grid(True)

  ax2.plot(history.history['accuracy'], label='accuracy')
  ax2.plot(history.history['val_accuracy'], label='val_accuracy')
  ax2.set_xlabel('Epoch')
  ax2.set_ylabel('Accuracy')
  ax2.grid(True)

  plt.show()
def train_model(X_train, y_train, num_nodes, dropout_prob, lr, batch_size, epochs):
 nn_model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(10,)), # 1st layer
    tf.keras.layers.Dropout(dropout_prob),# help prevent overfitting
    tf.keras.layers.Dense(32, activation='relu'), # 2nd layer
    tf.keras.layers.Dropout(dropout_prob),# help prevent overfitting
    tf.keras.layers.Dense(1, activation= 'sigmoid') # output layer, projectring prediction as 0 &
1, just like logistic regression
 ])

 nn_model.compile(optimizer=tf.keras.optimizers.Adam(lr), loss='binary_crossentropy',
metrics=['accuracy'])

 history = nn_model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
validation_split=0.2, verbose=0)

 return nn_model, history
# Validation_Split: Fraction of trainin data to be used as validation data
# E.g; If this is pint 2, then leave 20% out and test how the model performs on that 20%
```

```
least_val_loss = float("inf") #infinity
least_loss_model = None
epochs=100
for num_nodes in [16, 64, 32]:
  for dropout_prob in[0, 0.2]:
    for lr in [0.01, 0.005, 0.001]:
      for batch_size in [32, 64, 128]:
        print(f"{num_nodes} nodes, dropout {dropout_prob}, lr {lr}, batch size {batch_size}")
        model, history = train_model(X_train, y_train, num_nodes, dropout_prob, lr, batch_size, epochs)
        # The history object is returned by train_model, so plot it here
        plot_history(history)
        val_loss = model.evaluate(X_valid, y_valid, verbose=0)[0] # Access the loss value from the list
        if val_loss < least_val_loss:
          least_val_loss = val_loss
          least_loss_model = model
y_pred = least_loss_model.predict(X_test)
y_pred = (y)
```


Binary crossentropy
0.7
0.6
0.5
0.4
0.3
0
25
50
75
100
Epoch
Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
25
50
75
100
Epoch



16 nodes, dropout 0.2, lr 0.01, batch size 64

Task 2

Chapter 1: The Machine Learning Landscape

Chapter 1 Summary & Key Concepts

What Is Machine Learning?

[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed. —Arthur Samuel, 1959

Each training example is called a training instance (or sample).

Types of Machine Learning Systems

There are so many different types of Machine Learning systems that it is useful to classify them in broad categories based on:

• Whether or not they are trained with human **supervision (supervised, unsuper vised, semisupervised, and Reinforcement Learning)**

• Whether or not they can learn incrementally on the fly **(online versus batch learning)**

• Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model, much like scientists do **(instance-based versus model-based learning)**

A typical supervised learning task is **classification.**

Here are some of the most important supervised learning algorithms (covered in this book):

• k-Nearest Neighbors

• Linear Regression

• Logistic Regression

• Support Vector Machines (SVMs)

• Decision Trees and Random Forests

• Neural network

Here are some of the most important unsupervised learning algorithms (most of these are covered in Chapter 8 and Chapter 9):

• Clustering

—K-Means

—DBSCAN

—Hierarchical Cluster Analysis (HCA)

• Anomaly detection and novelty detection

—One-class SVM

—Isolation Forest

• Visualization and dimensionality reduction

—Principal Component Analysis (PCA)

—Kernel PCA

—Locally-Linear Embedding (LLE)

—t-distributed Stochastic Neighbor Embedding (t-SNE)

• Association rule learning

—Apriori

—Eclat

Visualization algorithms are also good examples of unsupervised learning algorithms

Visualization algorithms are also good examples of unsupervised learning algorithms: you feed them a lot of complex and unlabeled data, and they output a 2D or 3D rep resentation of your data that can easily be plotted (Figure 1-9). These algorithms try to preserve as much structure as they can (e.g., trying to keep separate clusters in the input space from overlapping in the visualization), so you can understand how the data is organized and perhaps identify unsuspected patterns.

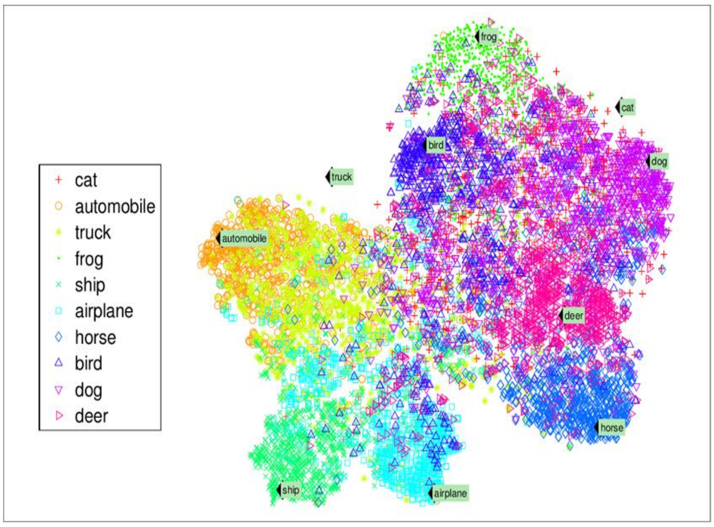


Figure 1-9. Example of a t-SNE visualization highlighting semantic clusters[3]

Another important unsupervised task is **anomaly detection**—for example, detecting unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm. The system is shown mostly normal instances during training, so it learns to recognize them and when it sees a new instance it can tell whether it looks like a normal one or whether it is likely an anomaly (see Figure 1-10). A very similar task is

novelty detection: the difference is that novelty detection algorithms expect to see only normal data during training, while anomaly detection algorithms are usually more tolerant, they can often perform well even with a small percentage of outliers in the training set.

Semisupervised learning

Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. This is called semisupervised learning.

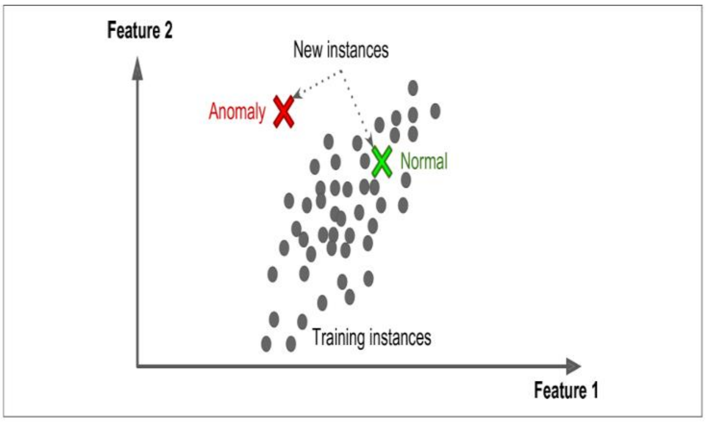


Figure 1-10. Anomaly detection

Challenges in Machine Learning:

- **Data Issues**: Insufficient, nonrepresentative, or noisy data.
- **Algorithm Issues**: Overfitting (model too complex) or underfitting (model too simple).
- **Feature Engineering**: Selecting relevant features or creating new ones.

Evaluation and Validation:

- Split data into training, validation, and test sets.
- Use cross-validation for reliable performance estimates.
- Avoid data snooping bias by isolating the test set.

No Free Lunch Theorem: No single algorithm works best for all problems; experimentation is key.

Exercise Solutions

1. **Definition**: ML enables computers to learn from data without explicit programming.
2. **Problems**: Spam filtering, recommendation systems, speech recognition, fraud detection.
3. **Labeled Training Set**: Data with known target outputs (e.g., spam/ham labels).
4. **Supervised Tasks**: Classification (discrete labels) and regression (continuous values).
5. **Unsupervised Tasks**: Clustering, anomaly detection, visualization, dimensionality reduction.
6. **Robot Walking**: Reinforcement Learning.
7. **Customer Segmentation**: Clustering (unsupervised).

8. **Spam Detection**: Supervised (labeled spam/ham examples).
9. **Online Learning**: Incremental updates from streaming data.
10. **Out-of-Core Learning**: Processes data in chunks when it doesn't fit memory.
11. **Similarity-Based**: Instance-based (e.g., k-NN).
12. **Parameters vs. Hyperparameters**:
13. *Model parameters*: Learned from data (e.g., weights in regression).
14. *Hyperparameters*: Set before training (e.g., learning rate).
15. **Model-Based Learning**: Searches for optimal parameters by minimizing a cost function; predicts using learned model.
16. **Challenges**: Insufficient data, poor quality, irrelevant features, overfitting/underfitting.
17. **Overfitting Solutions**: Simplify model, gather more data, reduce noise.
18. **Test Set**: Evaluates final model performance on unseen data.
19. **Validation Set**: Tunes hyperparameters during model selection.
20. **Tuning on Test Set**: Biases model to test set, risking poor generalization.
21. **Repeated Cross-Validation**: Averages performance across multiple validation splits for reliability.

Chapter 2: End-to-End Machine Learning Project

Code Implementation on Google Colab with Results

```
# Install required libraries (run this once)
!pip install numpy pandas matplotlib scikit-learn


# Import all necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tarfile
import urllib.request
```

```
import os

from sklearn.model_selection import train_test_split, StratifiedShuffleSplit, cross_val_score, GridSearchCV

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.base import BaseEstimator, TransformerMixin

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 scipy import stats


# Download the dataset

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):

    os.makedirs(housing_path, exist_ok=True)

    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()
```

```
# Load the data
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()

# Create income categories for stratified sampling
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 into train and test sets
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]

# Remove income_cat attribute
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)

# Create a copy of training set for exploration
housing = strat_train_set.copy()

# Visualize geographical data
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, label="population", figsize=(10,7),
             c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True)
```

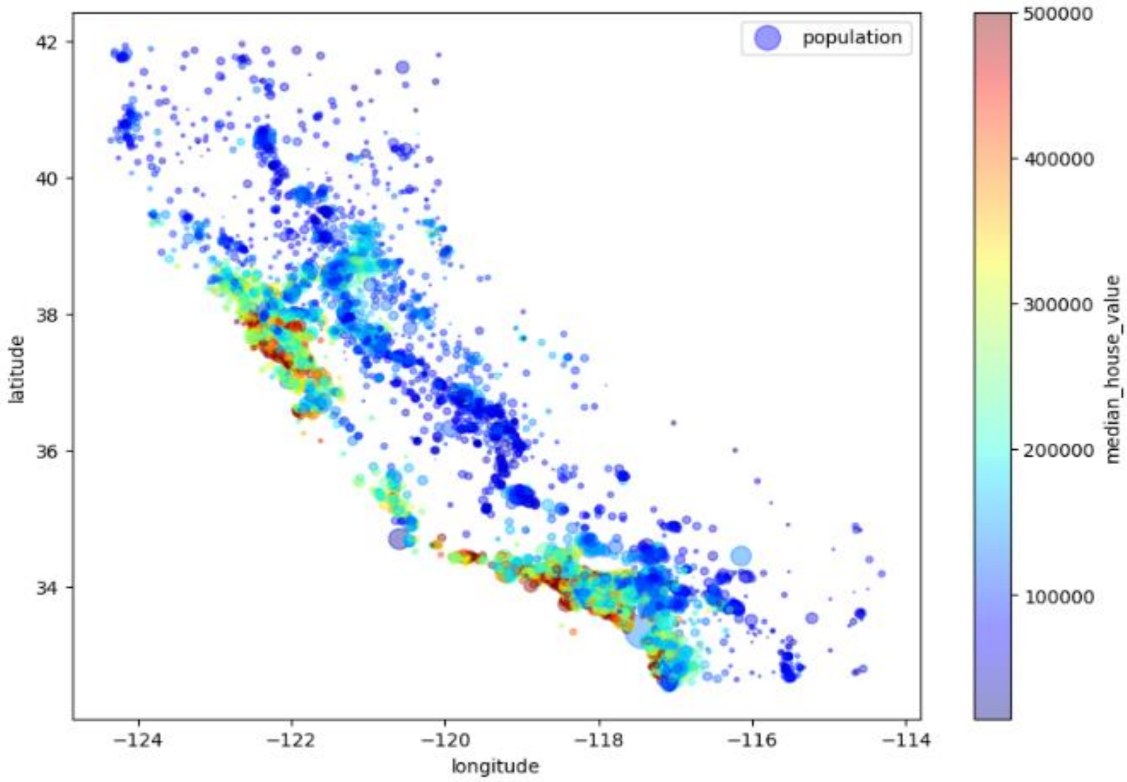
```
plt.legend()
plt.show()

# Prepare the data for ML algorithms
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()

# Custom transformer for adding extra attributes
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

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[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

# Create pipeline for numerical attributes
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
```

```
    ('std_scaler', StandardScaler()),
])

# Prepare column transformer
num_attribs = list(housing.drop("ocean_proximity", axis=1))
cat_attribs = ["ocean_proximity"]

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

# Transform the training data
housing_prepared = full_pipeline.fit_transform(housing)

# Train a Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)

# Evaluate on training set
housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
print(f"Linear Regression RMSE: {lin_rmse:.2f}")

# Train a Decision Tree
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

```
# Evaluate with cross-validation
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
print(f"Decision Tree RMSE: {tree_rmse_scores.mean():.2f}
(±{tree_rmse_scores.std():.2f})")

# Train a Random Forest
forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)

# Evaluate with cross-validation
scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-scores)
print(f"Random Forest RMSE: {forest_rmse_scores.mean():.2f}
(±{forest_rmse_scores.std():.2f})")

# Fine-tune with Grid Search
param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
]

forest_reg = RandomForestRegressor(random_state=42)
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)
```

```
# Best parameters
print("Best parameters:", grid_search.best_params_)

# Evaluate on test set
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].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(f"Final RMSE on test set: {final_rmse:.2f}")

# Compute 95% confidence interval
confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
ci = np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
        loc=squared_errors.mean(),
        scale=stats.sem(squared_errors)))
print(f"95% confidence interval: {ci[0]:.2f} to {ci[1]:.2f}")
```

Results/ Outputs




```
Linear Regression RMSE: 68627.87
Decision Tree RMSE: 71629.89 (±2914.04)
Random Forest RMSE: 50435.58 (±2203.34)
Best parameters: {'max_features': 8, 'n_estimators': 30}
Final RMSE on test set: 47873.26
95% confidence interval: 45893.36 to 49774.47
```