Main function

```
from fastapi import FastAPI, HTTPException, status, Response, Request, Form
from fastapi.responses import HTMLResponse
from fastapi.templating import Jinja2Templates
from pydantic import BaseModel
import pickle
import uvicorn


app = FastAPI()
templates = Jinja2Templates(directory="templates")
pickle_in = open("classifier.pkl", "rb")
classifier = pickle.load(pickle_in)


class Prostate(BaseModel):
    radius: int
    texture: int
    perimeter: int
    area: int
    smoothness: float
    compactness: float
    symmetry: float
    fractal_dimension: float


# main api function that get triggered when receving a request
@app.post("/prostates", status_code=status.HTTP_201_CREATED)
def create_heart(request: Request, Radius: str = Form(...), Texture: str =
Form(...), Perimeter: str = Form(...), Area: str = Form(...), Smoothness: str =
Form(...), Compactness: str = Form(...), Symmetry: str = Form(...),
Fractal_dimension: str = Form(...)):
    prediction = classifier.predict([[Radius, Area,
                                      Smoothness, Compactness, Symmetry]])
    if prediction[0] == 0:
        return templates.TemplateResponse("result.html", {"request": request,
"result": "Malignant Tumor"})
    else:
        return templates.TemplateResponse("result.html", {"request": request,
"result": "Benign Tumor"})
```

```
if __name__ == '__main__':
    uvicorn.run(app, host='0.0.0.0', port=8000)
```

Prostate model code

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import os
from sklearn.metrics import accuracy_score
import pickle

# loading the dataset
data = pd.read_csv('./Prostate_Cancer.csv')

# getting the first 10 lines of the dataset
data.head()

# descripting the dataset
data.describe()

# droping the id column
data = data.drop(['id'], axis=1)

# replace Malignant  with 0 and Benign  with 1
data['diagnosis_result'].replace({'M':0,'B':1},inplace=True)

# getting the corolation matrix of the data
corr_metrics = data.corr()
corr_metrics.style.background_gradient()

# eliminate the features that have a low correlation like Fracture dimension,
Texture, preimeter
data = data.drop(['fractal_dimension', 'texture', 'perimeter'], axis=1)
y = data['diagnosis_result'] # Labels
```

```
# splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state=10)

# importing the random forest classifier
forest = RandomForestClassifier(n_estimators = 50)

# training the model with the imported data
forest.fit(X_train,y_train)
pred_forest = forest.predict(X_test)

# getting the classification report
class_rep_forest = classification_report(y_test, pred_forest)
print("Forest Classifier: \n", class_rep_forest)


# getting the model accurecy
score=accuracy_score(y_test,pred_forest)

# saving the model for future use
pickle_out = open("classifier.pkl","wb")
pickle.dump(forest, pickle_out)
pickle_out.close()
```