Breast Cancer Detection Kaidoko round 2 AI Developer Intern

by Ishan Sharma

```
we are firstly going to import all the necessary libraries at first. We may
also need additional libraries so we will add those during that period only
```

In [1]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

sns.set()
plt.style.use('ggplot')
```

In [2]:

```python
df = pd.read_csv("C:\\Users\\ishan\\Downloads\\breast-cancer.csv")
df.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothn
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

5 rows × 32 columns

In [3]:

```python
df.diagnosis.unique()
```

Out[3]:

```
array(['M', 'B'], dtype=object)
```

In [4]:

```python
# M is Malignint or cancerous
# B is Benign or non cancerous
```

DATA PREPROCESSING¶

```
in this we usually have a set of processes which includes describe, it
helps in looking the overview of the data points and help in looking at the
outliers or help in suspicion (11.7 in radius mean are not very common so
anything signicicantly less could be outlier or help us)
```

In [5]:
```
df.describe()
```

Out[5]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0

8 rows × 31 columns

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
```

In [7]:
```
df.corr()
```

Out[7]:

	id	radius_mean	texture_mean	perimeter_mean	area_r
id	1.000000	0.074626	0.099770	0.073159	0.09
radius_mean	0.074626	1.000000	0.323782	0.997855	0.98
texture_mean	0.099770	0.323782	1.000000	0.329533	0.32
perimeter_mean	0.073159	0.997855	0.329533	1.000000	0.98
area_mean	0.096893	0.987357	0.321086	0.986507	1.00
smoothness_mean	-0.012968	0.170581	-0.023389	0.207278	0.17
compactness_mean	0.000096	0.506124	0.236702	0.556936	0.49
concavity_mean	0.050080	0.676764	0.302418	0.716136	0.68
concave points_mean	0.044158	0.822529	0.293464	0.850977	0.82
symmetry_mean	-0.022114	0.147741	0.071401	0.183027	0.15

In [8]:
```
df.drop('id', axis=1, inplace=True)
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
```

no missing value in this which is a good thing

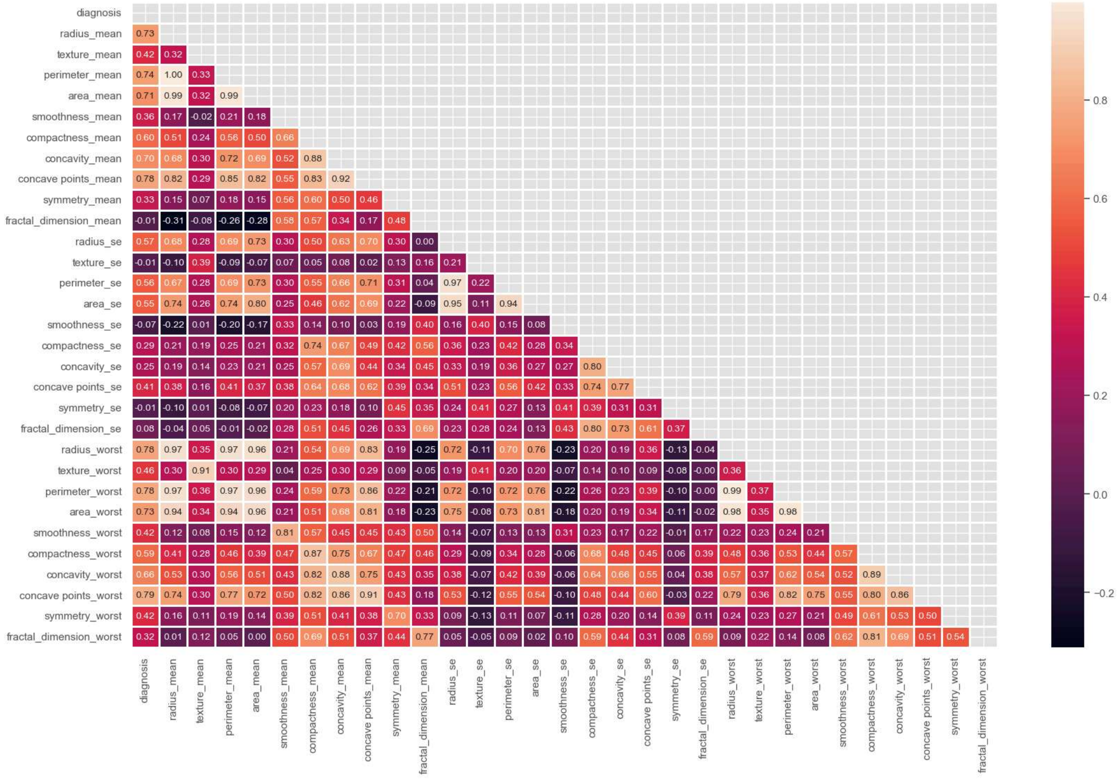
encoding categorical data

In [10]:

```
df['diagnosis'] = df['diagnosis'].apply(lambda val:1 if val=='M' else 0)
#we did [1,0] as it numericals are benefiary in model training and testing
```

In [11]:

```
plt.figure(figsize=(20,12))
corr=df.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(corr, mask=mask, linewidths=1, annot=True, fmt = ".2f")
plt.show()
```



In [12]:

```
corr_matrix = df.corr().abs()
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
tri_df = corr_matrix.mask(mask)

to_drop = [x for x in tri_df.columns if any(tri_df[x]>0.92)]

df = df.drop(to_drop, axis=1)

print(df.shape)
```

```
(569, 23)
```

In [13]:

```
df.head()
```

Out[13]:

	diagnosis	texture_mean	smoothness_mean	compactness_mean	concave points_mean	symmetr
0	1	10.38	0.11840	0.27760	0.14710	
1	1	17.77	0.08474	0.07864	0.07017	
2	1	21.25	0.10960	0.15990	0.12790	
3	1	20.38	0.14250	0.28390	0.10520	
4	1	14.34	0.10030	0.13280	0.10430	

5 rows × 23 columns

Buliding Model

In [14]:

```
X=df.drop('diagnosis', axis=1)
y=df['diagnosis']
```

In [15]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train ,y_test =train_test_split(X,y, test_size=0.2, rar
```

In [16]:

```
# scaling data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [17]:

```
X_train.shape
```

Out[17]:

```
(455, 22)
```

LogisticRegression

In [18]:

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

Out[18]:

```
▾ LogisticRegression

LogisticRegression()
```

In [19]:

```
y_pred = log_reg.predict(X_test)
```

In [20]:

```
y_pred
```

Out[20]:

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0], dtype=int64)
```

In [21]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, classificati
print(accuracy_score(y_train, log_reg.predict(X_train)))
log_reg_acc = accuracy_score(y_test, log_reg.predict(X_test))
print(log_reg_acc)
y_pred = log_reg.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.989010989010989
0.9649122807017544
[[66  1]
 [ 3 44]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        67
           1       0.98      0.94      0.96        47

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

Decision Tree

In [22]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

dtc = DecisionTreeClassifier()

parameters = {
    'criterion': ['gini', 'entropy'],
    'max_depth': range(2, 32, 1),
    'min_samples_leaf': range(1, 10, 1),
    'min_samples_split': range(2, 10, 1),
    'splitter': ['best', 'random']
}

grid_search_dt = GridSearchCV(dtc, parameters, cv=5, n_jobs=-1, verbose=1)
grid_search_dt.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 8640 candidates, totalling 43200 fits
```

Out[22]:

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
    ▸ DecisionTreeClassifier
```

In [23]:

```
grid_search_dt.best_params_
```

Out[23]:

```
{'criterion': 'gini',
 'max_depth': 6,
 'min_samples_leaf': 1,
 'min_samples_split': 7,
 'splitter': 'random'}
```

In [24]:

```
grid_search_dt.best_score_
```

Out[24]:

```
0.9626373626373625
```

In [25]:

```
dtc = DecisionTreeClassifier(criterion='entropy', max_depth=15, min_sample
```

In [26]:

```
dtc.fit(X_train, y_train)
```

Out[26]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=15, min_samples_l
eaf=4,
                       min_samples_split=5, splitter='random')
```

In [27]:

```python
print(accuracy_score(y_train, dtc.predict(X_train)))
dtc_acc = accuracy_score(y_test, dtc.predict(X_test))
print(dtc_acc)
y_pred = dtc.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.967032967032967
0.9473684210526315
[[67  0]
 [ 6 41]]
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        67
           1       1.00      0.87      0.93        47

    accuracy                           0.95       114
   macro avg       0.96      0.94      0.94       114
weighted avg       0.95      0.95      0.95       114
```

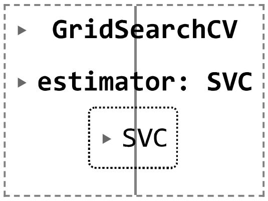
SVC

In [28]:

```python
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
svc= SVC(probability=True)

parameters = {
    'gamma': [0.0001, 0.001, 0.01, 0.1],
    'C':[0.01, 0.05, 0.5, 0.1, 1,10, 15,20]
}
grid_search = GridSearchCV(svc, parameters)
grid_search.fit(X_train, y_train)
```

Out[28]:



In [29]:

```python
grid_search.best_params_
```

Out[29]:

```
{'C': 15, 'gamma': 0.01}
```

In [30]:
```
grid_search.best_score_
```

Out[30]:
```
0.9802197802197803
```

In [31]:
```
svc = SVC(C=15, gamma=0.01, probability=True)
svc.fit(X_train, y_train)
```

Out[31]:
```
▾ SVC
SVC(C=15, gamma=0.01, probability=True)
```

In [32]:
```
y_pred = svc.predict(X_test)
```

In [33]:
```
print(accuracy_score(y_train, svc.predict(X_train)))
svc_acc = accuracy_score(y_test, svc.predict(X_test))
print(svc_acc)
y_pred = svc.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.989010989010989
0.9824561403508771
[[67  0]
 [ 2 45]]
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        67
           1       1.00      0.96      0.98        47

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

RandomForestClassifier

In [34]:

```
from sklearn.ensemble import RandomForestClassifier

rand_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 10, r
rand_clf.fit(X_train, y_train)
```

Out[34]:

```
                    RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=10, max_features=
0.5,
                       min_samples_leaf=2, min_samples_split=3,
                       n_estimators=130)
```

In [35]:

```
y_pred = rand_clf.predict(X_test)
```

In [36]:

```
print(accuracy_score(y_train, rand_clf.predict(X_train)))
rand_clf_acc = accuracy_score(y_test, rand_clf.predict(X_test))
print(rand_clf_acc)
y_pred = rand_clf.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9978021978021978
0.9824561403508771
[[66  1]
 [ 1 46]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        67
           1       0.98      0.98      0.98        47

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

XGBClassifier

In [37]:

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\ishan\anaconda3\lib\s
ite-packages (2.1.0)
Requirement already satisfied: numpy in c:\users\ishan\anaconda3\lib\sit
e-packages (from xgboost) (1.23.5)
Requirement already satisfied: scipy in c:\users\ishan\anaconda3\lib\sit
e-packages (from xgboost) (1.10.0)
Note: you may need to restart the kernel to use updated packages.
```

In [38]:

```
from xgboost import XGBClassifier

xgb = XGBClassifier(objective = 'binary:logistic', learning_rate = 0.01, r

xgb.fit(X_train, y_train)
```

Out[38]:

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rou
nds=None,
              enable_categorical=False, eval_metric=None, feature_ty
pes=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.01, max_
bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
```

In [39]:

```
print(accuracy_score(y_train, xgb.predict(X_train)))
xgb_acc = accuracy_score(y_test, xgb.predict(X_test))
print(xgb_acc)
y_pred = xgb.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
0.9934065934065934
0.956140350877193
[[65  2]
 [ 3 44]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.96        67
           1       0.96      0.94      0.95        47

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

Model Evaluation

In [40]:

```
models = pd.DataFrame({
    'Model': ['Logistic Regression', 'Decision Tree Classifier', 'SVM', 'R
    'Score': [100*round(log_reg_acc,4), 100*round(dtc_acc,4), 100*round(sv
               100*round(xgb_acc,4)]
})
models.sort_values(by = 'Score', ascending = False)
```

Out[40]:

	Model	Score
2	SVM	98.25
3	Random Forest Classifier	98.25
0	Logistic Regression	96.49
4	XgBoost	95.61
1	Decision Tree Classifier	94.74