Assignment 2

- Name: **Arnab Sen**
- Roll: **510519006**
- Date: **Aug 12, 2022**

(i) Download data

Data downloaded and stored at drive.

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
BASE_PATH = '/content/drive/MyDrive/Colab_Notebooks/ML_DRIVE/Assign_2/dataset'
```

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.naive_bayes import BernoulliNB
from random import randint
from statistics import mean
import matplotlib.pyplot as plt
```

In [ ]:
```
dataset = pd.read_csv(f"{BASE_PATH}/data.csv")
print("Dataset shape:", dataset.shape)
print("Dataset columns:", dataset.columns)
```

```
Dataset shape: (569, 33)
Dataset columns: Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'p
erimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_s
e',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

In [ ]:
```
dataset = dataset.drop(columns = ['id', 'Unnamed: 32'])
dataset
```

Out[ ]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.11100	
565	M	20.13	28.25	131.20	1261.0	0.09780	
566	M	16.60	28.08	108.30	858.1	0.08455	
567	M	20.60	29.33	140.10	1265.0	0.11780	
568	B	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

(ii) Implement Logistic regression

Implement Logistic regression using scikit-learn package in python after splitting the dataset 80:10:10 percent (use seed = 5 for splitting).

In [ ]:
```
def train_validate_test_split(df, train_percent=.8, validate_percent=.1, seed=N
    np.random.seed(seed)
    perm = np.random.permutation(df.index)
    m = len(df.index)
    train_end = int(train_percent * m)
    validate_end = int(validate_percent * m) + train_end
    train = df.iloc[perm[:train_end]]
    validate = df.iloc[perm[train_end:validate_end]]
    test = df.iloc[perm[validate_end:]]
    return train, validate, test
```

In [ ]:
```
train_df, validation_df, test_df = train_validate_test_split(dataset, train_pe
print("Shape of train:", train_df.shape)
print("Shape of validation:", validation_df.shape)
print("Shape of test:", test_df.shape)
```

```
Shape of train: (455, 31)
Shape of validation: (56, 31)
Shape of test: (58, 31)
```

In [ ]:
```
train_df
```

Out[ ]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
28	M	15.300	25.27	102.40	732.4	0.10820	
163	B	12.340	22.22	79.85	464.5	0.10120	
123	B	14.500	10.89	94.28	640.7	0.11010	
361	B	13.300	21.57	85.24	546.1	0.08582	
549	B	10.820	24.21	68.89	361.6	0.08192	
...	...	...	...	...	...	...	
266	B	10.600	18.95	69.28	346.4	0.09688	
470	B	9.667	18.49	61.49	289.1	0.08946	
473	B	12.270	29.97	77.42	465.4	0.07699	
169	B	14.970	16.95	96.22	685.9	0.09855	
36	M	14.250	21.72	93.63	633.0	0.09823	

455 rows × 31 columns

In [ ]:

```
validation_df
```

Out[ ]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
243	B	13.750	23.77	88.54	590.0	0.08043	
82	M	25.220	24.91	171.50	1878.0	0.10630	
260	M	20.310	27.06	132.90	1288.0	0.10000	
433	M	18.820	21.97	123.70	1110.0	0.10180	
348	B	11.470	16.03	73.02	402.7	0.09076	
372	M	21.370	15.10	141.30	1386.0	0.10010	
314	B	8.597	18.60	54.09	221.2	0.10740	
22	M	15.340	14.26	102.50	704.4	0.10730	
337	M	18.770	21.43	122.90	1092.0	0.09116	
308	B	13.500	12.71	85.69	566.2	0.07376	
550	B	10.860	21.48	68.51	360.5	0.07431	
239	M	17.460	39.28	113.40	920.6	0.09812	
220	B	13.650	13.16	87.88	568.9	0.09646	
13	M	15.850	23.95	103.70	782.7	0.08401	
485	B	12.450	16.41	82.85	476.7	0.09514	
160	B	11.750	20.18	76.10	419.8	0.10890	
72	M	17.200	24.52	114.20	929.4	0.10710	
177	M	16.460	20.11	109.30	832.9	0.09831	
276	B	11.330	14.16	71.79	396.6	0.09379	
367	B	12.210	18.02	78.31	458.4	0.09231	
35	M	16.740	21.59	110.10	869.5	0.09610	
326	B	14.110	12.88	90.03	616.5	0.09309	
207	M	17.010	20.26	109.70	904.3	0.08772	
476	B	14.200	20.53	92.41	618.4	0.08931	
225	B	14.340	13.47	92.51	641.2	0.09906	
120	B	11.410	10.82	73.34	403.3	0.09373	
216	B	11.890	18.35	77.32	432.2	0.09363	
29	M	17.570	15.05	115.00	955.1	0.09847	
347	B	14.760	14.74	94.87	668.7	0.08875	
188	B	11.810	17.39	75.27	428.9	0.10070	
544	B	13.870	20.70	89.77	584.8	0.09578	
325	B	12.670	17.30	81.25	489.9	0.10280	
180	M	27.220	21.87	182.10	2250.0	0.10940	
406	B	16.140	14.86	104.30	800.0	0.09495	
233	M	20.510	27.81	134.40	1319.0	0.09159	
508	B	16.300	15.70	104.70	819.8	0.09427	

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
459	B	9.755	28.20	61.68	290.9	0.07984	
505	B	9.676	13.14	64.12	272.5	0.12550	
165	B	14.970	19.76	95.50	690.2	0.08421	
126	M	13.610	24.69	87.76	572.6	0.09258	
556	B	10.160	19.59	64.73	311.7	0.10030	
474	B	10.880	15.62	70.41	358.9	0.10070	
264	M	17.190	22.07	111.60	928.3	0.09726	
64	M	12.680	23.84	82.69	499.0	0.11220	
16	M	14.680	20.13	94.74	684.5	0.09867	
214	M	14.190	23.81	92.87	610.7	0.09463	
532	B	13.680	16.33	87.76	575.5	0.09277	
487	M	19.440	18.82	128.10	1167.0	0.10890	
384	B	13.280	13.72	85.79	541.8	0.08363	
338	B	10.050	17.53	64.41	310.8	0.10070	
522	B	11.260	19.83	71.30	388.1	0.08511	
175	B	8.671	14.45	54.42	227.2	0.09138	
398	B	11.060	14.83	70.31	378.2	0.07741	
518	B	12.880	18.22	84.45	493.1	0.12180	
172	M	15.460	11.89	102.50	736.9	0.12570	
332	B	11.220	19.86	71.94	387.3	0.10540	

56 rows × 31 columns

In [ ]:
```
test_df
```

Out[ ]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
32	M	17.020	23.98	112.80	899.3	0.11970	
31	M	11.840	18.70	77.93	440.6	0.11090	
235	B	14.030	21.25	89.79	603.4	0.09070	
374	B	13.690	16.07	87.84	579.1	0.08302	
397	B	12.800	17.46	83.05	508.3	0.08044	
408	M	17.990	20.66	117.80	991.7	0.10360	
418	B	12.700	12.17	80.88	495.0	0.08785	
389	M	19.550	23.21	128.90	1174.0	0.10100	
552	B	12.770	29.43	81.35	507.9	0.08276	
265	M	20.730	31.12	135.70	1419.0	0.09469	
14	M	13.730	22.61	93.60	578.3	0.11310	
94	M	15.060	19.83	100.30	705.6	0.10390	
391	B	8.734	16.84	55.27	234.3	0.10390	
78	M	20.180	23.97	143.70	1245.0	0.12860	
387	B	13.880	16.16	88.37	596.6	0.07026	
183	B	11.410	14.92	73.53	402.0	0.09059	
449	M	21.100	20.52	138.10	1384.0	0.09684	
380	B	11.270	12.96	73.16	386.3	0.12370	
491	B	17.850	13.23	114.60	992.1	0.07838	
190	M	14.220	23.12	94.37	609.9	0.10750	
364	B	13.400	16.95	85.48	552.4	0.07937	
135	M	12.770	22.47	81.72	506.3	0.09055	
245	B	10.480	19.86	66.72	337.7	0.10700	
274	M	17.930	24.48	115.20	998.9	0.08855	
147	B	14.950	18.77	97.84	689.5	0.08138	
105	M	13.110	15.56	87.21	530.2	0.13980	
294	B	12.720	13.78	81.78	492.1	0.09667	
324	B	12.200	15.21	78.01	457.9	0.08673	
539	B	7.691	25.44	48.34	170.4	0.08668	
110	B	9.777	16.99	62.50	290.2	0.10370	
5	M	12.450	15.70	82.57	477.1	0.12780	
144	B	10.750	14.97	68.26	355.3	0.07793	
103	B	9.876	19.40	63.95	298.3	0.10050	
210	M	20.580	22.14	134.70	1290.0	0.09090	
446	M	17.750	28.03	117.30	981.6	0.09997	
41	M	10.950	21.35	71.90	371.1	0.12270	

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
362	B	12.760	18.84	81.87	496.6	0.09676	
377	B	13.460	28.21	85.89	562.1	0.07517	
254	M	19.450	19.33	126.50	1169.0	0.10350	
146	M	11.800	16.58	78.99	432.0	0.10910	
86	M	14.480	21.46	94.25	648.2	0.09444	
542	B	14.740	25.42	94.70	668.6	0.08275	
431	B	12.400	17.68	81.47	467.8	0.10540	
65	M	14.780	23.94	97.40	668.3	0.11720	
205	M	15.120	16.68	98.78	716.6	0.08876	
44	M	13.170	21.81	85.42	531.5	0.09714	
27	M	18.610	20.25	122.10	1094.0	0.09440	
80	B	11.450	20.97	73.81	401.5	0.11020	
437	B	14.040	15.98	89.78	611.2	0.08458	
113	B	10.510	20.19	68.64	334.2	0.11220	
204	B	12.470	18.60	81.09	481.9	0.09965	
519	B	12.750	16.70	82.51	493.8	0.11250	
411	B	11.040	16.83	70.92	373.2	0.10770	
8	M	13.000	21.82	87.50	519.8	0.12730	
73	M	13.800	15.79	90.43	584.1	0.10070	
400	M	17.910	21.02	124.40	994.0	0.12300	
118	M	15.780	22.91	105.70	782.6	0.11550	
206	B	9.876	17.27	62.92	295.4	0.10890	

58 rows × 31 columns

In [ ]:

```
y_test, y_train, y_valid = test_df['diagnosis'], train_df['diagnosis'], valida
X_test, X_train, X_valid = test_df.drop('diagnosis', axis=1), train_df.drop('d
```

(iii) Train Logistic Regression Model

In [ ]:

```
def train_model_with_solver(X_train, y_train, X_valid, y_valid, solver, penalty
  lr = LogisticRegression(solver = solver, max_iter = 10000, penalty=penalty, (
  lr.fit(X_train, y_train)
  score = lr.score(X_valid, y_valid)

  return {
      "solver": solver,
      "score": score,
      "coefs": lr.coef_.tolist()[0],
      "penalty": penalty,
```

```
        "inv_of_regularization": C
    }
```

In [ ]:
```
def display_table(models, columns):
    headers = ['solver', 'accuracy', 'penalty', 'inv_of_regularization'] + colum
    data = [[model['solver'], model['score'], model["penalty"], model["inv_of_re

    return pd.DataFrame(
        columns = headers,
        data = data
    )
```

In [ ]:
```
newton_cg_model = train_model_with_solver(X_train, y_train, X_valid, y_valid, '
lbfgs_model = train_model_with_solver(X_train, y_train, X_valid, y_valid, "lbf
liblinear_model = train_model_with_solver(X_train, y_train, X_valid, y_valid, '
```

In [ ]:
```
display_table([newton_cg_model, lbfgs_model, liblinear_model], X_train.columns
```

Out[ ]:

	solver	accuracy	penalty	inv_of_regularization	radius_mean	texture_mean	perimeter_mean
0	newton-cg	0.964286	l2	1.0	-0.482746	-0.114460	0.151515
1	lbfgs	0.964286	l2	1.0	-0.406884	-0.112863	0.155643
2	liblinear	0.946429	l2	1.0	-1.653319	-0.091261	-0.029303

3 rows × 34 columns

(iv) Use ‘l1’, ‘l2’, ‘none’ penality to train the Logistic regression model.

In [ ]:
```
l1_model = train_model_with_solver(X_train, y_train, X_valid, y_valid, "saga",
l2_model = train_model_with_solver(X_train, y_train, X_valid, y_valid, "saga",
none_model = train_model_with_solver(X_train, y_train, X_valid, y_valid, "saga"
```

In [ ]:
```
display_table([l1_model, l2_model, none_model], X_train.columns)
```

Out[ ]:

	solver	accuracy	penalty	inv_of_regularization	radius_mean	texture_mean	perimeter_mean
0	saga	0.910714	l1	1.0	-0.014208	-0.009116	-0.076412
1	saga	0.910714	l2	1.0	-0.014422	-0.009283	-0.076457
2	saga	0.910714	none	1.0	-0.014426	-0.009281	-0.076474

3 rows × 34 columns

(v) Vary the l1 penalty over the range (0.1, 0.25, 0.75, 0.9)

compare the coefficients of the features.

In [ ]:
```
penalties = [0.1, 0.25, 0.75, 0.9]
models = [train_model_with_solver(X_train, y_train, X_valid, y_valid, "saga", p
display_table(models, X_train.columns)
```

Out[ ]:

	solver	accuracy	penalty	inv_of_regularization	radius_mean	texture_mean	perimeter_mean	a
0	saga	0.910714	l1	0.10	-0.012256	-0.007591	-0.075876	
1	saga	0.910714	l1	0.25	-0.013558	-0.008598	-0.076237	
2	saga	0.910714	l1	0.75	-0.014136	-0.009062	-0.076391	
3	saga	0.910714	l1	0.90	-0.014186	-0.009092	-0.076415	

4 rows × 34 columns

(vi) Estimate the average accuracy of the Naive Bayes algorithm using 5-fold cross-validation

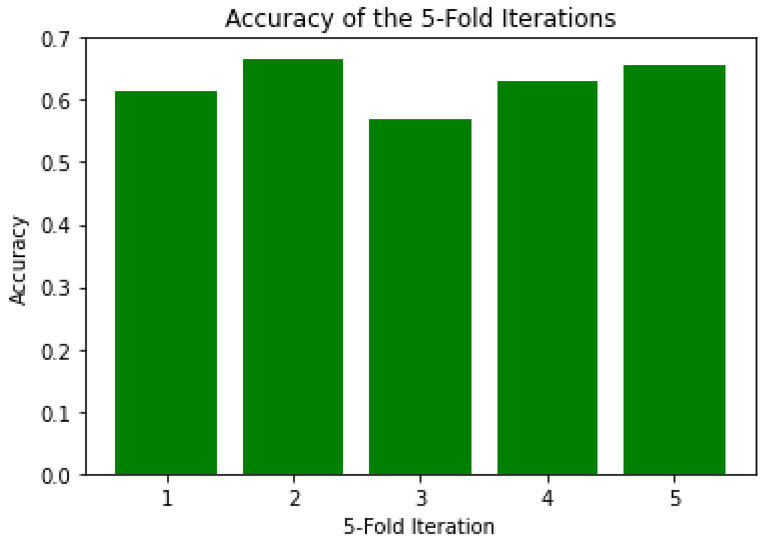
Use scikit-learn package in python. Plot the bar graph using matplotlib.

In [ ]:
```
X = dataset.drop('diagnosis', axis=1)
y = dataset['diagnosis']
folds = KFold(n_splits=5, shuffle=True)
nb_accuracy = []
for train_ids, test_ids in folds.split(X):
  X_train = X.iloc[train_ids]
  y_train = y.iloc[train_ids]
  X_test = X.iloc[test_ids]
  y_test = y.iloc[test_ids]

  naive_bayes_model = BernoulliNB()
  naive_bayes_model.fit(X_train, y_train)

  accuracy = naive_bayes_model.score(X_test, y_test)
  nb_accuracy.append(accuracy)

print("Avg accuracy = ", mean(nb_accuracy))
```

```
Avg accuracy =  0.6274646793976091
```

In [ ]:
```
plt.xlabel('5-Fold Iteration')
plt.ylabel('Accuracy')
plt.title('Accuracy of the 5-Fold Iterations')
plt.bar([x for x in range(1,6)],nb_accuracy, color='green')
plt.plot()
```

Out[ ]:
```
[]
```


Accuracy of the 5-Fold Iterations
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Accuracy
1
2
3
4
5
5-Fold Iteration