Part 1: Data preprocessing

Dataset source: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

1. Importing the libraries
2. Importing the dataset
3. Dealing with missing data
4. Encoding categorical variables
5. Splitting the dataset into train and test set
6. Feature scaling

Importing the libraries and dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv('/content/data.csv')
```

Data exploration

```
dataset.head(25)
```

```
{"type":"dataframe","variable_name":"dataset"}
```

```
dataset.shape
```

```
(569, 33)
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 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
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
dataset.select_dtypes(include='object').columns
len(dataset.select_dtypes(include='object').columns)  # There is only
one column with 'object' data type
```

```
1
```

```
dataset = dataset.drop(columns='Unnamed: 32')
```

```
dataset.head()
```

{"type":"dataframe","variable_name":"dataset"}

```
# Statistical summary
dataset.describe()
```

{"type":"dataframe"}

```
dataset.columns
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean',
'perimeter_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_se',
       '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'],
      dtype='object')
```

Dealing with the missing data

```
# Check if there are any null values
dataset.isnull().values.any()
```

```
False
```

```
# Check how many null values
dataset.isnull().values.sum()
```

```
0
```

Encoding the categorical data

```
dataset.select_dtypes(include='object').columns
```

```
Index(['diagnosis'], dtype='object')
```

```
dataset['diagnosis'].unique() # Unique values in the data
```

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

```
dataset['diagnosis'].nunique() # Number of unique values in the data
```

```
2
```

```
dataset.head()
```

{"type":"dataframe","variable_name":"dataset"}

```
dataset.shape
```

```
(569, 32)
```

```
dataset.head()
```

{"type":"dataframe","variable_name":"dataset"}

```
dataset = pd.read_csv('/content/data.csv')
```

```
dataset.head()
```

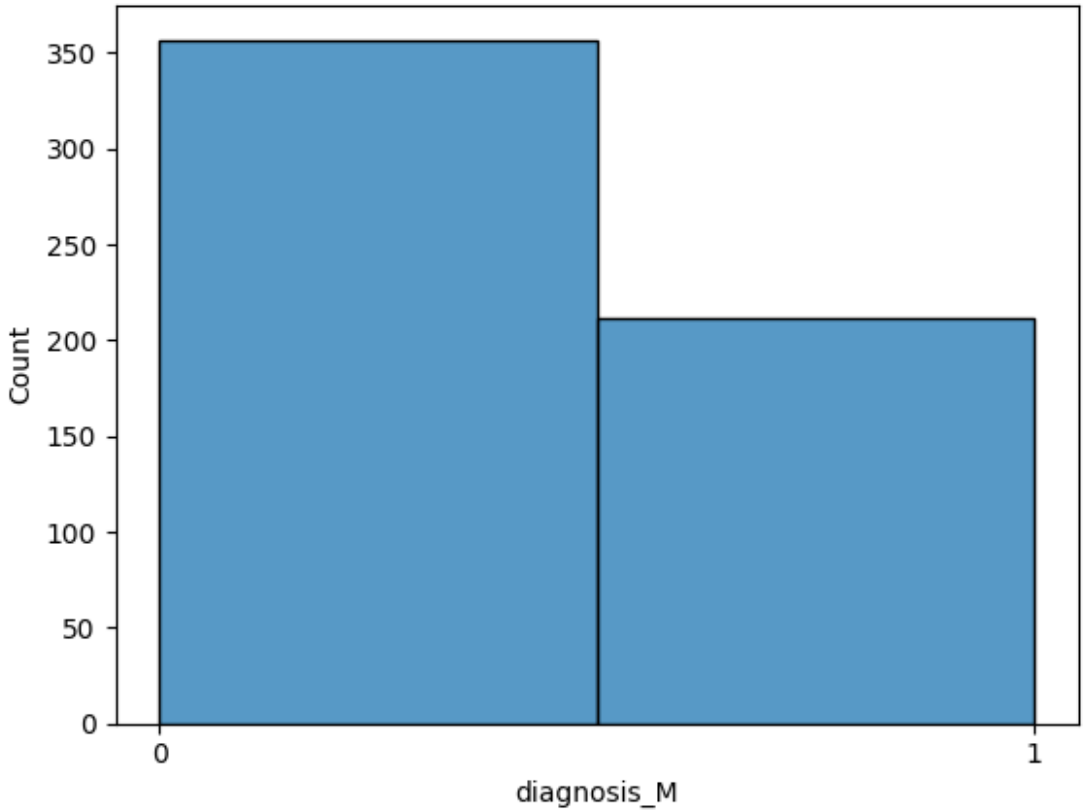
{"type":"dataframe","variable_name":"dataset"}

```
dataset = dataset.drop(columns='Unnamed: 32')

dataset = pd.get_dummies(data=dataset, drop_first=True, dtype=int)

dataset.head()
```

{"type":"dataframe","variable_name":"dataset"}

Countplot

```
sns.histplot(data=dataset['diagnosis_M'], bins=2)
plt.xticks(ticks=[0, 1], labels=['0', '1'])
plt.show()
```




```
# benign (B) values
(dataset.diagnosis_M == 0).sum()
```

```
357
```

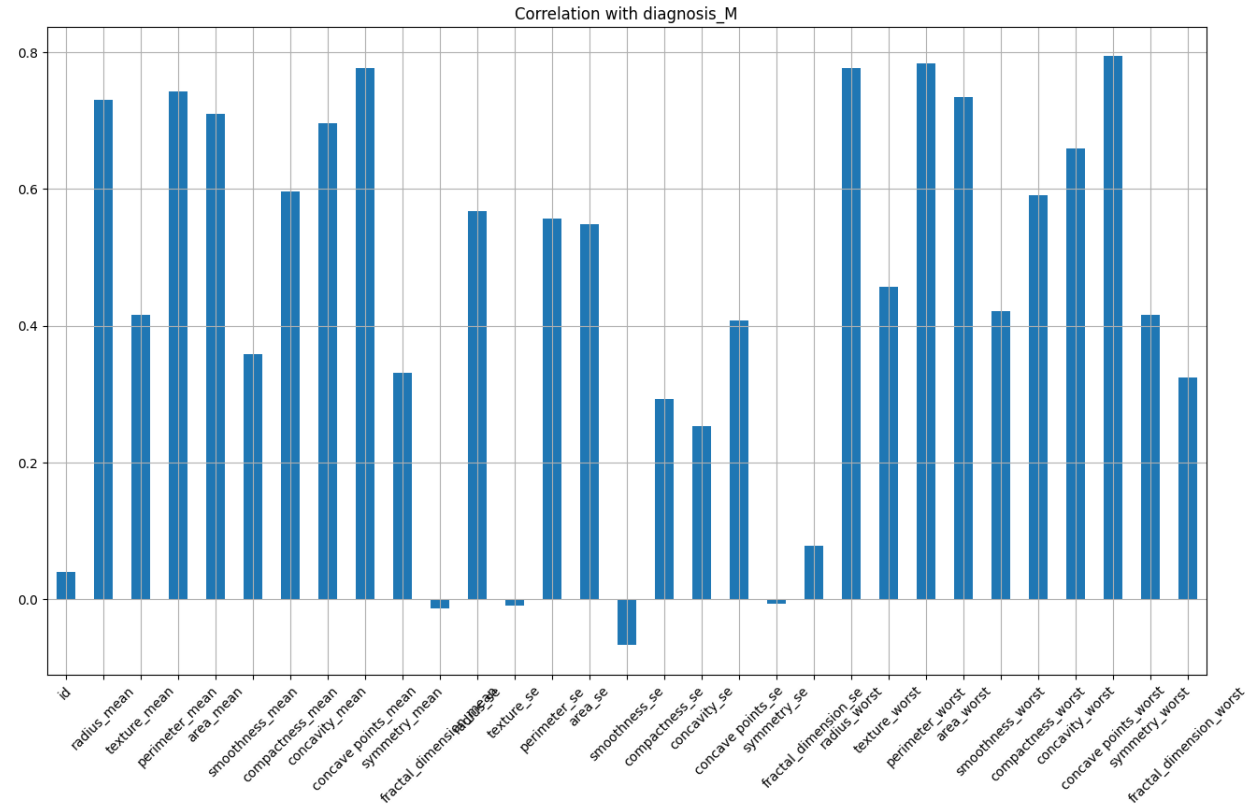
```
# malignant (M) values
(dataset.diagnosis_M == 1).sum()
```

```
212
```

Correlation matrix and Heatmap

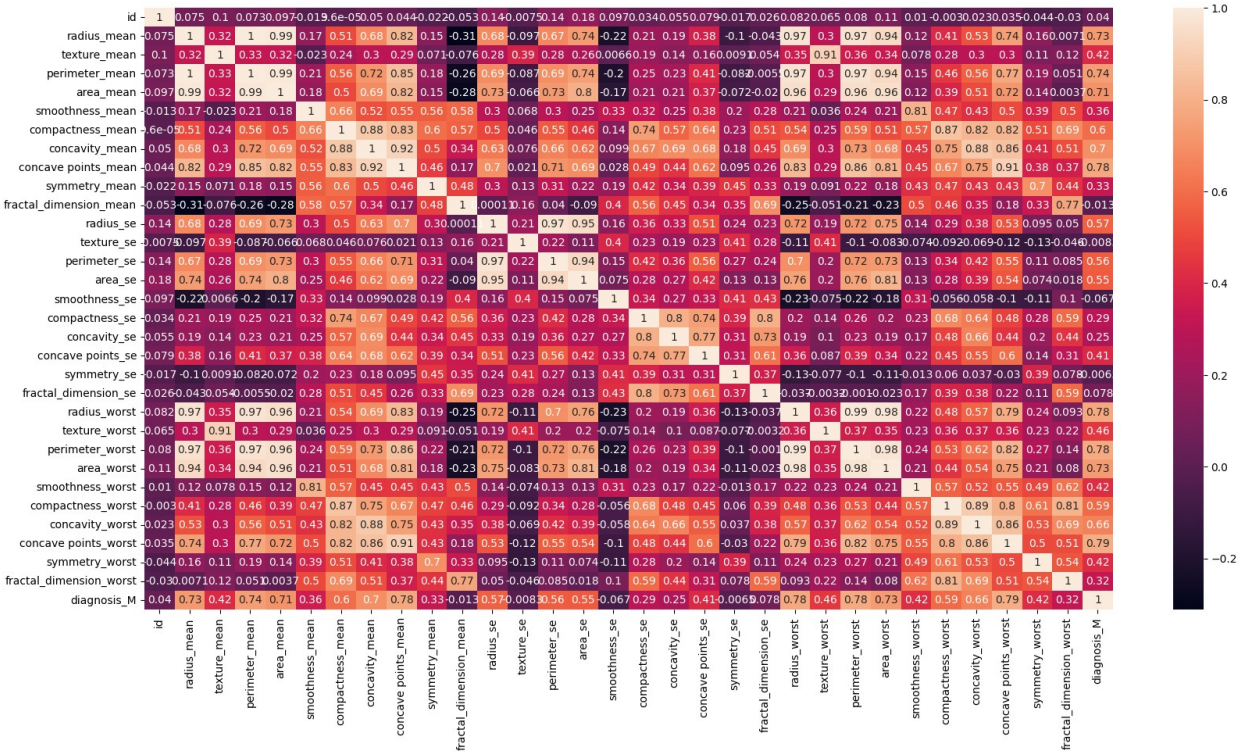
```
dataset_2 = dataset.drop(columns='diagnosis_M')

dataset_2.corrwith(dataset['diagnosis_M']).plot.bar(
    figsize=(16,9), title = 'Correlation with diagnosis_M',
    rot = 45, grid = True
)   # How much each variable in the feature matrix is correlated with
the target variable
```

```
<Axes: title={'center': 'Correlation with diagnosis_M'}>
```




```
# Create Correlation Matrix
corr = dataset.corr()

# Check the correlation between variables

plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True)
```

```
<Axes: >
```



Splitting the dataset into train and test set

```
dataset.head()
```


```
{"type":"dataframe","variable_name":"dataset"}
```


```
# matrix of features / independent variables
x = dataset.iloc[:, 1:-1].values # : -> all rows , 1:-1 -> start fromm column 1(after id one) and end beforoe last column which is our target column
```

```
x.shape
```

```
(569, 30)
```

```
# dependent variable
y = dataset.iloc[:, -1].values
```

```
y.shape
```

```
(569,)
```

```
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, random_state=0)
```

```
x_train.shape
```

```
(455, 30)
```

```
x_test.shape
```

```
(114, 30)
```

```
y_train.shape
```

```
(455,)
```

```
y_test.shape
```

```
(114,)
```

Feature scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
x_train
```

```
array([[-1.15036482, -0.39064196, -1.12855021, ..., -0.75798367,
        -0.01614761, -0.38503402],
       [-0.93798972,  0.68051405, -0.94820146, ..., -0.60687023,
         0.09669004, -0.38615797],
       [ 0.574121  , -1.03333557,  0.51394098, ..., -0.02371948,
        -0.20050207, -0.75144254],
       ...,
       [-1.32422924, -0.20048168, -1.31754581, ..., -0.97974953,
        -0.71542314, -0.11978123],
       [-1.24380987, -0.2245526 , -1.28007609, ..., -1.75401433,
        -1.58157125, -1.00601779],
       [-0.73694129,  1.14989702, -0.71226578, ..., -0.27460457,
        -1.25895095,  0.21515662]])
```

```
x_test
```

```
array([[-0.20175604,  0.3290786 , -0.13086754, ...,  1.3893291 ,
         1.08203284,  1.54029664],
       [-0.25555773,  1.46763319, -0.31780437, ..., -0.83369364,
        -0.73131577, -0.87732522],
       [-0.02619262, -0.8407682 , -0.09175081, ..., -0.49483785,
        -1.22080864, -0.92115937],
       ...,
       [ 1.71811488,  0.09318356,  1.7286186 , ...,  1.57630515,
         0.20317063, -0.15406178],
       [ 1.18859296,  0.34352115,  1.19333694, ...,  0.56019755,
         0.26991966, -0.27320074],
       [ 0.26263752, -0.58080224,  0.28459338, ..., -0.19383705,
        -1.15564888,  0.11231497]])
```

Part 2: Building the model

1) Logistic regression

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=0)
clf.fit(x_train, y_train)
```

```
LogisticRegression(random_state=0)
```

```
y_pred = clf.predict(x_test)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,
f1_score, precision_score, recall_score

acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)


results = pd.DataFrame([['Logistic Regression', acc, f1, prec, rec]],
               columns = ['Model', 'Accuracy', 'F1 Score',
'Precision', 'Recall'])

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[65  2]
 [ 2 45]]
```

```
results
```


```
{"summary":"{\n  \"name\": \"results\",\n  \"rows\": 1,\n  \"fields\":
[\n    {\n      \"column\": \"Model\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          \"Logistic Regression\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Accuracy\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": null,\n
\"min\": 0.9649122807017544,\n        \"max\": 0.9649122807017544,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
0.9649122807017544\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"F1
Score\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": null,\n        \"min\": 0.9574468085106383,\n        \"max\":
0.9574468085106383,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          0.9574468085106383\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Precision\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
null,\n        \"min\": 0.9574468085106383,\n        \"max\":
0.9574468085106383,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          0.9574468085106383\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Recall\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": null,\n
\"min\": 0.9574468085106383,\n        \"max\": 0.9574468085106383,\n
\"num_unique_values\": 1,\n        \"samples\": [\n
0.9574468085106383\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"results"}
```


Cross validation

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=clf, X=x_train, y=y_train,
cv=10)

print("Accuracy is {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation is {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy is 97.81 %
Standard Deviation is 1.98 %
```

2) Random forest

```
from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(random_state=0)
classifier_rf.fit(x_train, y_train)
```

```
RandomForestClassifier(random_state=0)
```

```
y_pred = classifier_rf.predict(x_test)

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['Random Forest', acc, prec, rec, f1]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall',
'F1 Score'])


results = pd.concat([results, model_results],ignore_index=True)
results
```


```
{"summary":"{\n  \"name\": \"results\",\n  \"rows\": 2,\n  \"fields\":
[\n    {\n      \"column\": \"Model\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 2,\n
```

\"samples\": [\n   \"Random Forest\",\n   \"Logistic Regression\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Accuracy\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0,\n   \"min\": 0.9649122807017544,\n   \"max\": 0.9649122807017544,\n   \"num_unique_values\": 1,\n   \"samples\": [\n   0.9649122807017544\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"F1 Score\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.0006268677138178524,\n   \"min\": 0.9574468085106383,\n   \"max\": 0.9583333333333334,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0.9583333333333334\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Precision\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.013202601646123123,\n   \"min\": 0.9387755102040817,\n   \"max\": 0.9574468085106383,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0.9387755102040817\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Recall\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.015044825131628614,\n   \"min\": 0.9574468085106383,\n   \"max\": 0.9787234042553191,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0.9787234042553191\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"results"}

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[64  3]
 [ 1 46]]
```

Cross validation

```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=classifier_rf, X=x_train,
y=y_train, cv=10)

print("Accuracy is {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation is {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy is 96.05 %
Standard Deviation is 3.07 %
```

Part 3: Randomized Search to find the best parameters (Logistic regression)

```
from sklearn.model_selection import RandomizedSearchCV

parameters = [
              {'penalty':['l1', 'l2', 'elasticnet', 'none'],
               'C':[0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2],
               'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag',
'saga']
               }
              ]

parameters
```

```
[{'penalty': ['l1', 'l2', 'elasticnet', 'none'],
  'C': [0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2],
  'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}]
```

```
randomized_search = RandomizedSearchCV(estimator = clf,
param_distributions = parameters,
                                       n_iter = 10, scoring='roc_auc',
n_jobs = -1, cv = 10, verbose=3)

# cv: cross-validation
# n_jobs = -1:
# Number of jobs to run in parallel. -1 means using all processors

randomized_search.fit(x_train, y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/
_validation.py:540: FitFailedWarning:
40 fits failed out of a total of 100.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by
setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------
----------
10 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line
```

```
1473, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 1194, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logisti
c.py", line 67, in _check_solver
    raise ValueError(
ValueError: Solver sag supports only 'l2' or None penalties, got
elasticnet penalty.

--------------------------------------------------------------------
----------
30 fits failed with the following error:
Traceback (most recent call last):
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_vali
dation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line
1466, in wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line
666, in _validate_params
    validate_parameter_constraints(
  File
"/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validati
on.py", line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'penalty'
parameter of LogisticRegression must be a str among {'l2',
'elasticnet', 'l1'} or None. Got 'none' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_searc
h.py:1103: UserWarning: One or more of the test scores are non-finite:
[       nan 0.99629817        nan        nan 0.99587982        nan
 0.99467546 0.99548682 0.99568966 0.99528398]
  warnings.warn(

RandomizedSearchCV(cv=10,
estimator=LogisticRegression(random_state=0),
                   n_jobs=-1,
                   param_distributions=[{'C': [0.25, 0.5, 0.75, 1,
1.25, 1.5,
                                               1.75, 2],
                                         'penalty': ['l1', 'l2',
'elasticnet',
```

```
                                                          'none'],
                                                'solver': ['newton-cg',
'lbfgs',
                                                           'liblinear',
'sag',
                                                           'saga']}],
                    scoring='roc_auc', verbose=3)

randomized_search.best_estimator_

LogisticRegression(C=1.25, penalty='l1', random_state=0,
solver='liblinear')

randomized_search.best_params_

{'solver': 'liblinear', 'penalty': 'l1', 'C': 1.25}

randomized_search.best_score_

0.9962981744421906
```

Part 4: Final Model (Logistic regression)

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(C=1, class_weight=None, dual=False,
fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l1',
                    random_state=0, solver='saga', tol=0.0001,
verbose=0,
                    warm_start=False)
classifier.fit(x_train, y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in
version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this
warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(

LogisticRegression(C=1, multi_class='auto', penalty='l1',
random_state=0,
                    solver='saga')

y_pred = classifier.predict(x_test)
```

```
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_results = pd.DataFrame([['Final Logistic regression', acc, prec,
rec, f1]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall',
'F1 Score'])

results = pd.concat([results, model_results],ignore_index=True)
results
```


```
{"summary":"{\n  \"name\": \"results\",\n  \"rows\": 3,\n  \"fields\":
[\n    {\n      \"column\": \"Model\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"Logistic Regression\",\n          \"Random
Forest\",\n          \"Final Logistic regression\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Accuracy\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.005064476045523049,\n        \"min\": 0.956140350877193,\n
\"max\": 0.9649122807017544,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.956140350877193,\n
0.9649122807017544\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"F1
Score\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0067427437924985996,\n        \"min\": 0.946236559139785,\n
\"max\": 0.9583333333333334,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.9574468085106383,\n
0.9583333333333334\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Precision\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.010523004758758695,\n        \"min\":
0.9387755102040817,\n        \"max\": 0.9574468085106383,\n
\"num_unique_values\": 3,\n        \"samples\": [\n
0.9574468085106383,\n          0.9387755102040817\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Recall\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.021276595744680826,\n        \"min\": 0.9361702127659575,\n
\"max\": 0.9787234042553191,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          0.9574468085106383,\n
0.9787234042553191\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"results"}
```


```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator=classifier, X=x_train,
y=y_train, cv=10)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in
version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this
warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5
and will be removed in 1.7. From then on, it will always use
'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5
and will be removed in 1.7. From then on, it will always use
'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5
and will be removed in 1.7. From then on, it will always use
'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5
and will be removed in 1.7. From then on, it will always use
'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5
and will be removed in 1.7. From then on, it will always use
'multinomial'. Leave it to its default value to avoid this warning.
```

```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5
and will be removed in 1.7. From then on, it will always use
'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5
and will be removed in 1.7. From then on, it will always use
'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5
and will be removed in 1.7. From then on, it will always use
'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic
.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5
and will be removed in 1.7. From then on, it will always use
'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_sag.py:3
49: ConvergenceWarning: The max_iter was reached which means the coef_
did not converge
  warnings.warn(
```

```python
print("Accuracy is {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation is {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy is 98.02 %
Standard Deviation is 2.08 %
```

The final logistic regression model has accuracy of 98.02%.

The standard deviation is 2.08%

Part 5: Predicting a single observation

```
dataset.head()
```

```
{"type":"dataframe","variable_name":"dataset"}
```

```
dataset.shape
```

```
(569, 32)
```

```
single_obs = [[20.34, 17.50, 131.40, 1320.15, 0.084, 0.079, 0.087,
0.069, 0.180, 0.057,
  0.542, 0.730, 3.400, 74.50, 0.005, 0.013, 0.019, 0.014, 0.014,
0.003,
  24.80, 23.00, 159.00, 1950, 0.124, 0.185, 0.240, 0.190, 0.280,
0.090]]


single_obs
```

```
[[20.34,
  17.5,
  131.4,
  1320.15,
  0.084,
  0.079,
  0.087,
  0.069,
  0.18,
  0.057,
  0.542,
  0.73,
  3.4,
  74.5,
  0.005,
  0.013,
  0.019,
  0.014,
  0.014,
  0.003,
  24.8,
```

```
  23.0,
  159.0,
  1950,
  0.124,
  0.185,
  0.24,
  0.19,
  0.28,
  0.09]]

classifier.predict(sc.transform(single_obs))

array([1])

# '1' means the patient is malignant.
```