import libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
np.random.seed(42)
```

Getting the Data Ready

```
diabetes_data = pd.read_csv("diabetes-dataset.csv")
diabetes_data.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	2	138	62	35	0	33.6	
1	0	84	82	31	125	38.2	
2	0	145	0	0	0	44.2	
3	0	135	68	42	250	42.3	
4	1	139	62	41	480	40.7	

```
diabetes_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               2000 non-null   int64
 1   Glucose                   2000 non-null   int64
 2   BloodPressure             2000 non-null   int64
 3   SkinThickness             2000 non-null   int64
 4   Insulin                   2000 non-null   int64
 5   BMI                       2000 non-null   float64
 6   DiabetesPedigreeFunction  2000 non-null   float64
 7   Age                       2000 non-null   int64
 8   Outcome                   2000 non-null   int64
dtypes: float64(2), int64(7)
memory usage: 140.8 KB
```

```
diabetes_data.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BI
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.0000
mean	3.703500	121.182500	69.145500	20.935000	80.254000	32.1930
std	3.306063	32.068636	19.188315	16.103243	111.180534	8.1499
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	1.000000	99.000000	63.500000	0.000000	0.000000	27.3750
50%	3.000000	117.000000	72.000000	23.000000	40.000000	32.3000
75%	6.000000	141.000000	80.000000	32.000000	130.000000	36.8000

- rows where Glucose is 0
- rows where BloodPressure is 0
- rows where Insulin is 0
- rows where BMI is 0 which is not possible, so Lets replace those rows with there column mean.

```
# First lets check how many 0 values are in these columns
featureList = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
diabetes_data[featureList].isin([0]).sum()
```

```
Glucose           13
BloodPressure     90
SkinThickness    573
Insulin          956
BMI               28
dtype: int64
```

- for Glucose, BloodPressure, SkinThickness and BMI due to low number of '0' values we can replace it with their mean. But we will have to check it replacing in Insulin will affect accuracy beacuse there are around ~50% data that is '0'.

Data Cleaning

```
#cleaning of data and replaced with corresponding mean value

# For Glucose
diabetes_data["Glucose"] = diabetes_data["Glucose"].replace({ 0 : diabetes_data["Glucose"]

# For BloodPressure
diabetes_data["BloodPressure"] = diabetes_data["BloodPressure"].replace({ 0 : diabetes_dat

# For SkinThickness
diabetes_data["SkinThickness"] = diabetes_data["SkinThickness"].replace({ 0 : diabetes_dat

# For BMI
```

```
diabetes_data["BMI"] = diabetes_data["BMI"].replace({ 0 : diabetes_data["BMI"].mean()})

# Lets checkif it worked
diabetes_data[featureList].isin([0]).sum()
```

```
Glucose            0
BloodPressure      0
SkinThickness      0
Insulin          956
BMI                0
dtype: int64
```

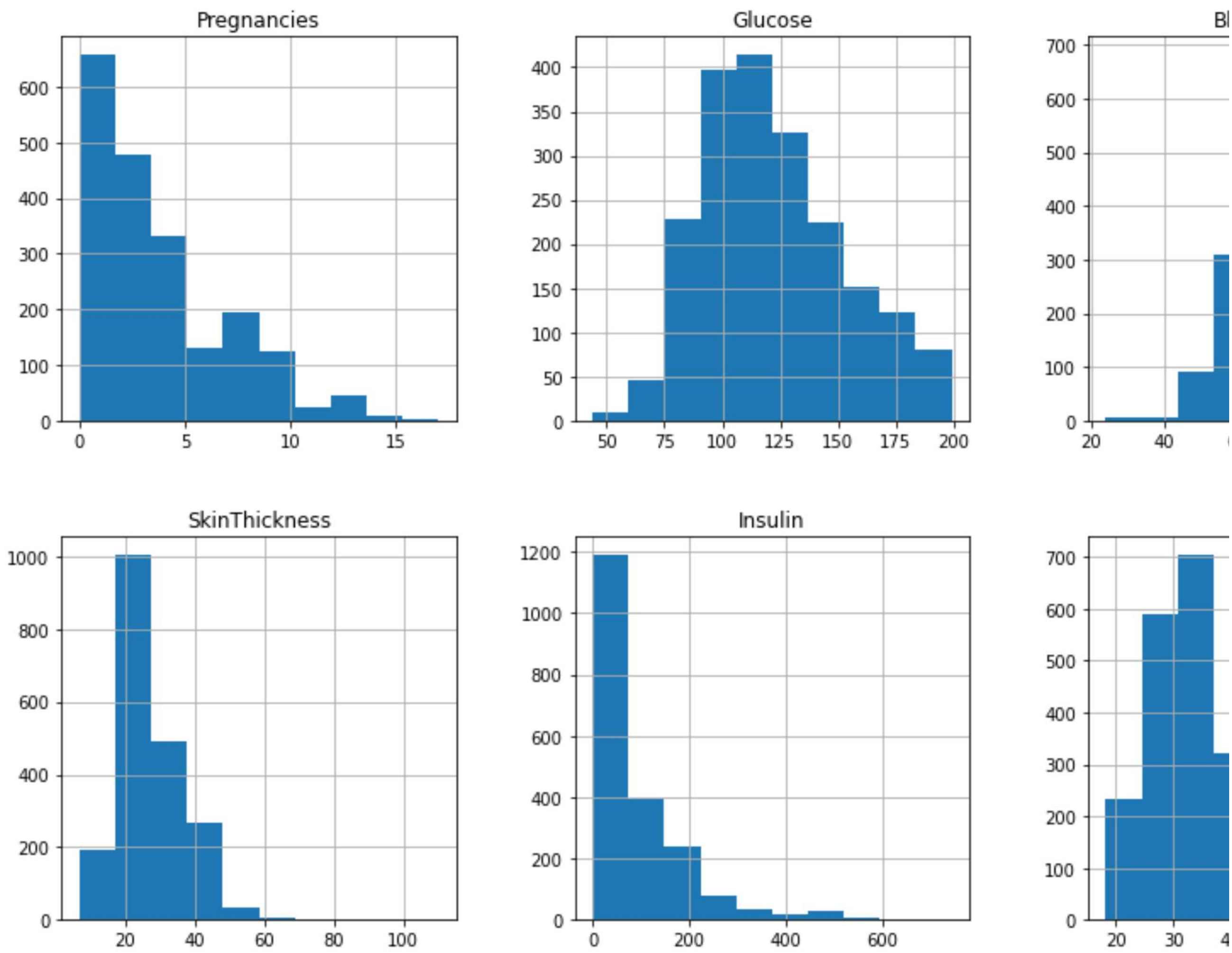
```
diabetes_data_copy = diabetes_data.copy(deep = True)
diabetes_data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diabetes

# Showing the Count of NANs
print(diabetes_data_copy.isnull().sum())
```

```
Pregnancies                   0
Glucose                       0
BloodPressure                 0
SkinThickness                 0
Insulin                     956
BMI                           0
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

Data Visualization

```
p = diabetes_data.hist(figsize = (15,15))
```



- Because there are 956/2000 rows of Insulin where data is '0' we will train 2 models where we drop Insulin

```
diabetes_data_idroped = diabetes_data.drop(diabetes_data[diabetes_data["Insulin"] == 0].in
diabetes_data_idroped.shape
```

```
(1044, 9)
```

```
# Splitting data into X & y

X = diabetes_data_idroped.drop(["Outcome"], axis=1)
y = diabetes_data_idroped["Outcome"]

X.shape , y.shape
```

```
((1044, 8), (1044,))
```

```
# Spliting data into training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Fitting the data to the right model

```
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.model_selection import KFold, cross_val_score
for model in [
    DummyClassifier,
    DecisionTreeClassifier,
    KNeighborsClassifier,
    GaussianNB,
    SVC,
    RandomForestClassifier]:

    cls = model()
    kf = KFold(n_splits = 5)
    score = cross_val_score(cls, X_train, y_train, cv = kf, scoring="roc_auc")

    print(f"{model.__name__:22}  AUC: \t {score.mean():.3f} STD: {score.std():.2f}")
```

```
DummyClassifier         AUC:     0.500 STD: 0.00
DecisionTreeClassifier  AUC:     0.942 STD: 0.02
KNeighborsClassifier    AUC:     0.866 STD: 0.03
GaussianNB              AUC:     0.833 STD: 0.03
SVC                     AUC:     0.846 STD: 0.02
RandomForestClassifier  AUC:     0.990 STD: 0.00
```

```
# Fitting the modle model
cls = RandomForestClassifier()

# Fitting the model
cls.fit(X_train, y_train)

# Prediction
y_preds = cls.predict(X_test)
```

Evaluating the model

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       185
           1       0.97      0.97      0.97        76

    accuracy                           0.98       261
```

```
   macro avg       0.98      0.98      0.98       261
weighted avg       0.98      0.98      0.98       261
```

```
from sklearn.metrics import roc_auc_score
print(roc_auc_score(y_test, y_preds))
```

```
0.9814366998577526
```

Tuning the parameters

```
from sklearn.model_selection import RandomizedSearchCV

# Define a grid of hyperparameters
grid = {"n_estimators": [10, 100, 200, 500, 1000, 1200],
        "max_depth": [None, 5, 10, 20, 30],
        "max_features": ["auto", "sqrt"],
        "min_samples_split": [2, 4, 6],
        "min_samples_leaf": [1, 2, 4]}

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Set n_jobs to -1 to use all cores (NOTE: n_jobs=-1 is broken as of 8 Dec 2019, using n_j
clf = RandomForestClassifier(n_jobs=1)

# Setup RandomizedSearchCV
rs_clf = RandomizedSearchCV(estimator=clf,
                            param_distributions=grid,
                            n_iter=10, # try 10 models total
                            cv=5, # 5-fold cross-validation
                            verbose=2) # print out results


# Fit the RandomizedSearchCV version of clf
rs_clf.fit(X_train, y_train);

# Find the best hyperparameters
print(rs_clf.best_params_)

# Scoring automatically uses the best hyperparameters
rs_clf.score(X_test, y_test)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_split=2,
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_
```

```
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=6, n_
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=6, n_
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=6, n_
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=6, n_
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=6, n_
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_split=6, n_
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=4,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=4,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=4,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=4,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_split=4,
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_
[CV] END max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_split=4, n_
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=2,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=2,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=2,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=2,
[CV] END max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_split=2,
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_
[CV] END max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_
[CV] END max_depth=30, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=4, min_samples_split=4, n_e
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=4, min_samples_split=4, n_e
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=4, min_samples_split=4, n_e
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=4, min_samples_split=4, n_e
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=4, min_samples_split=4, n_e
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=4, n_
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=4, n_
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=4, n_
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=4, n_
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=4, n_
{'n_estimators': 100, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features':
0.9846743295019157
```

```
print(f"Final Score: {rs_clf.score(X_test, y_test)}")
```

```
Final Score: 0.9846743295019157
```