Import Data and Libraries

```
import pandas as pd
import numpy as np
import pyarrow.feather as feather

df = feather.read_feather('/content/diabetes.feather')
```

1) Prepare the x and y objects

```
df = pd.get_dummies(df, columns=['race', 'gender', 'age', 'admission_type_id', 'discharge_disposition_id',
                                 'admission_source_id', 'diag_1', 'diag_2', 'diag_3'])
```

```
# drop the 'readmitted' to create x  and create y
x = df.drop(columns=['readmitted'], axis = 1)
```

```
y = df["readmitted"]
y = (y == ">30").astype(int)
```

2) Split the data into two sets

```
# Split the data into training and test sets with 80/20 split and stratify on y
from sklearn.linear_model import LogisticRegression
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 = 12)
```

3) Build initial models

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score

# Instantiate the models
mod_rf = RandomForestClassifier(n_estimators=500, random_state=1)
mod_boost = GradientBoostingClassifier(n_estimators=500, random_state=1)
```

```
# Perform 10-fold cross-validation and calculate F1-score for each model
f1_rf = cross_val_score(mod_rf, x_train, y_train, cv=10, scoring='f1')
f1_boost = cross_val_score(mod_boost, x_train, y_train, cv=10, scoring='f1')

cv_f1_rf = f1_rf.mean()
cv_f1_boost = f1_boost.mean()

print('RF CV F1-score:', cv_f1_rf)
print('GB CV F1-score:', cv_f1_boost)
```

```
RF CV F1-score: 0.1944029337430245
GB CV F1-score: 0.1416202132362271
```

4) Build models with tuned hyperpamrameters

```
from sklearn.model_selection import RandomizedSearchCV

# Define the parameter grids for Random Forest and Gradient Boosting
params_rf = {
    'n_estimators': [100, 300, 500, 800, 1000],
    'max_depth': [5, 10, 15, 20, 25, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None],
    'bootstrap': [True, False]
}

params_boost = {
    'n_estimators': [100, 300, 500, 800, 1000],
    'learning_rate': [0.001, 0.01, 0.1, 0.5, 1],
    'max_depth': [3, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None]
```

```
}

# Perform random search on Random Forest and Gradient Boosting
rs_rf = RandomizedSearchCV(
    RandomForestClassifier(),
    params_rf,
    cv=10,
    n_iter=6,
    scoring='f1',
    random_state=1
)

rs_boost = RandomizedSearchCV(
    GradientBoostingClassifier(),
    params_boost,
    cv=10,
    n_iter=6,
    scoring='f1',
    random_state=1
)

# Fit the models and print the best F1-score
rs_rf.fit(x_train, y_train)
rs_boost.fit(x_train, y_train)

print('Best F1-score for Random Forest:', rs_rf.best_score_)
print('Best F1-score for Gradient Boosting:', rs_boost.best_score_)
```

```
Best F1-score for Random Forest: 0.16013333046961506
Best F1-score for Gradient Boosting: 0.24029735468412716
```

5) Build models that undersample negative cases

```
x_fold1, x_fold2, y_fold1, y_fold2 = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

# Assuming y_fold1 is the target variable containing 0s and 1s
pos_indx = np.where(y_fold1 == 1)[0] # get the indices where y_fold1 is 1
neg_indx = np.where(y_fold1 == 0)[0] # get the indices where y_fold1 is

m = int(np.floor(len(pos_indx) * 0.5))
np.random.seed(1)

sample_indx = np.random.choice(neg_indx, size=m, replace=False)

x_subsampled = pd.concat([x_fold1.iloc[pos_indx], x_fold1.iloc[sample_indx]])
y_subsampled = pd.concat([y_fold1.iloc[pos_indx], y_fold1.iloc[sample_indx]])
```

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# Train a Random Forest classifier with 500 decision trees
mod_rf_2 = RandomForestClassifier(n_estimators=500, random_state=1).fit(x_subsampled, y_subsampled)

# Train a Gradient Boosting classifier with 500 decision trees
mod_boost_2 = GradientBoostingClassifier(n_estimators=500, random_state=1).fit(x_subsampled, y_subsampled)
```

```
from sklearn.metrics import f1_score

def get_f1(y_true, y_pred):
    f1 = f1_score(y_true, y_pred)
    if not f1:
        return 0
    else:
        return f1

pred_mod_rf_2 = mod_rf_2.predict(x_fold2)
pred_mod_boost_2 = mod_boost_2.predict(x_fold2)
```

```
from sklearn.metrics import f1_score

# Calculate the F1 scores for mod_rf_2 and mod_boost_2
f1_mod_rf_2 = get_f1(y_fold2, pred_mod_rf_2)
f1_mod_boost_2 = get_f1(y_fold2, pred_mod_boost_2)

# Print the F1 scores
print("Random Forest F1 score:", f1_mod_rf_2)
print("Gradient Boosting F1 score:", f1_mod_boost_2)
```

```
Random Forest F1 score: 0.47626495659654333
Gradient Boosting F1 score: 0.47964628754383287
```

6) Make a dataframe of model specifications (We attempted steps 6 to 9 but stopped at 7.4 since they are optional)

```
df_n_estimators = pd.DataFrame({"n_estimators": np.arange(200, 1001, 200)})
df_min_samples_leaf = pd.DataFrame({"min_samples_leaf": np.arange(0.01, 0.1, 0.02)})
df_max_depth = pd.DataFrame({"max_depth": np.arange(1, 6, 1)})
df_subsample = pd.DataFrame({"subsample": [0.5, 1, 2]})
df_max_depth["id"] = 1
df_n_estimators["id"] = 1
df_min_samples_leaf["id"] = 1
df_subsample["id"] = 1
df_rf = pd.merge(df_n_estimators, df_min_samples_leaf, on = "id")
df_rf = pd.merge(df_rf, df_subsample, on = "id")
df_rf["estimator"] = "Random Forest"
df_rf["max_depth"] = None
df_rf = df_rf.drop("id", axis = 1)
df_boost = pd.merge(df_n_estimators, df_max_depth, on = "id")
df_boost = pd.merge(df_boost, df_subsample, on = "id")
df_boost["estimator"] = "Boosting"
df_boost["min_samples_leaf"] = None
df_boost = df_boost.drop("id", axis = 1)
df_models = pd.concat([df_rf, df_boost], ignore_index = True)
```

7) Select the best model using CV

```
n_models = df_models.shape[0]

n_models = len(df_models)
cv_rslt = np.zeros((n_models, 10))
```

```
n = np.ceil(len(y_train) / 10)
fold_vec = np.concatenate([np.arange(10)] * int(n))
fold_vec = fold_vec[0:len(y)]
np.random.seed(1)
fold_vec = np.random.permutation(fold_vec)
```

```
from sklearn.metrics import roc_auc_score
```

```
# Step 7.4
for i in range(10):
  indx_1fold = ...
  indx_9fold = ...
  x_1fold = ...
  y_1fold = ...
  x_9fold = ...
  y_9fold = ...


  for j in range(n_models):
    # Get the jth model specifics
    estimator = ...
    n_estimators = ...
    min_samples_leaf = ...
    subsample = ...
    max_depth = ...
    # Instantiate the model
    if estimator == "Random Forest":
      mod = ...
    if estimator == "Boosting":
      mod = ...
    # Subsample the negative cases, using the object "subsample"
    pos_indx = ...
    neg_indx = ...
    m = ...
    np.random.seed(i)
    sample_indx = ...
    x_subsampled = ...
    y_subsampled = ...
    mod.fit(..., ...)
    pred_mod = ...
    cv_rslt... = ...
```