0. Import data

[ ] ↳ *4 cells hidden*

Data Preprocessing

```
#Checking for missing values in dataset
#In the dataset missing values are represented as '?' sign
for col in df.columns:
    if df[col].dtype == object:
         print(col,df[col][df[col] == '?'].count())
```

```
race 0
gender 0
age 0
diag_1 0
diag_2 0
diag_3 0
max_glu_serum 0
A1Cresult 0
metformin 0
repaglinide 0
nateglinide 0
chlorpropamide 0
glimepiride 0
acetohexamide 0
glipizide 0
glyburide 0
tolbutamide 0
pioglitazone 0
rosiglitazone 0
acarbose 0
miglitol 0
troglitazone 0
tolazamide 0
insulin 0
glyburide-metformin 0
glipizide-metformin 0
glimepiride-pioglitazone 0
metformin-pioglitazone 0
change 0
diabetesMed 0
readmitted 0
```

```
# deal with missing value
df = df.replace('?', np.nan)
```

```
# You should combine the < 30 days and > 30 days values to “Yes”.
# you need to ensure that categorical variables are represented with (k-1) dummy variables that have values 0 or 1.
df = df.replace({'readmitted': '>30'}, {'readmitted': 'YES'})
df = df.replace({'readmitted': '<30'}, {'readmitted': 'YES'})
df['readmitted'].value_counts()
```

```
NO     52338
YES    45715
Name: readmitted, dtype: int64
```

```
# Numerical data:
numerical_cols = ['time_in_hospital',   'num_lab_procedures',   'num_procedures',
                  'num_medications',    'number_outpatient',    'number_emergency',
                  'number_inpatient',   'number_diagnoses']
```

```
# race  gender  age  admission_type_id  discharge_disposition_id
# admission_source_id       max_glu_serum   A1Cresult
# metformin repaglinide nateglinide chlorpropamide  glimepiride acetohexamide   glipizide
# glyburide tolbutamide pioglitazone    rosiglitazone   acarbose    miglitol    troglitazone    tolazamide  insulin
# glyburide-metformin   glipizide-metformin glimepiride-pioglitazone    metformin-pioglitazone
# change    diabetesMed

# Categorical data:
category_cols = ['race', 'gender', 'age', 'admission_type_id', 'discharge_disposition_id',
                 'admission_source_id', 'max_glu_serum', 'A1Cresult', 'metformin',
                 'repaglinide', 'nateglinide',  'chlorpropamide',   'glimepiride',
                 'acetohexamide',   'glipizide', 'glyburide',   'tolbutamide',  'pioglitazone',
                 'rosiglitazone',   'acarbose', 'miglitol', 'troglitazone', 'tolazamide',
                 'insulin', 'glyburide-metformin',  'glipizide-metformin',  'glimepiride-pioglitazone',
                 'metformin-pioglitazone', 'change',    'diabetesMed']

# encounter_id and patient_nbr: these are just identifiers and not useful variables
# diag1, diag2, diag3 – are categorical and have a lot of values. We will use number_diagnoses to capture some of this information.
```

Create dummy 1 by 1

[ ] ↳ 2 cells hidden

Get all dummies

```
term_adm_type = pd.get_dummies(df['admission_type_id'], prefix='adm_type_id')
term_dis_dispo = pd.get_dummies(df['discharge_disposition_id'], prefix='disch_dispo_id')
term_adm_source = pd.get_dummies(df['admission_source_id'], prefix='adm_source_id')

ids = term_adm_type.join(term_dis_dispo)
ids = ids.join(term_adm_source)


# Get dummies for all categorical data
cat_data = pd.get_dummies(df[category_cols]).drop(columns=['admission_type_id', 'discharge_disposition_id', 'admission_source_id'])
cat_data = ids.join(cat_data)


final_X = df[numerical_cols].join(cat_data)
y = df['readmitted']


final_X
```

	time_in_hospital	num_lab_procedures	num_procedures	num_medications	number_outpatient	number_emergency	number_inpatient	number_diagnoses
0	3	59	0	18	0	0	0	9
1	2	11	5	13	2	0	1	6
2	2	44	1	16	0	0	0	7
3	1	51	0	8	0	0	0	5
4	3	31	6	16	0	0	0	9
...	...	...	...	...	...	...	...	...
98048	3	51	0	16	0	0	0	9
98049	5	33	3	18	0	0	1	9
98050	1	53	0	9	1	0	0	13
98051	10	45	2	21	0	0	1	9
98052	6	13	3	3	0	0	0	9

98053 rows × 156 columns

Train Test Split

```
X_train, X_test, y_train, y_test = train_test_split(final_X, y, test_size=0.3, random_state=42)


X_train
```

Part 1. Logistic Regression

Perfrom GridSearchCV to find best hyperparemters

```
import warnings
warnings.filterwarnings('ignore')

# parameter grid
parameters = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3,3,7),
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}


logreg = LogisticRegression()
clf = GridSearchCV(logreg,                  # model
                   param_grid = parameters,   # hyperparameters
```

```
                   scoring='accuracy',        # metric for scoring
                   cv=5)                     # number of folds
clf.fit(X_train,y_train)

print("Tuned Hyperparameters :", clf.best_params_)
print("Accuracy :",clf.best_score_)
```

```
     Tuned Hyperparameters : {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
     Accuracy : 0.6322245830569451
```

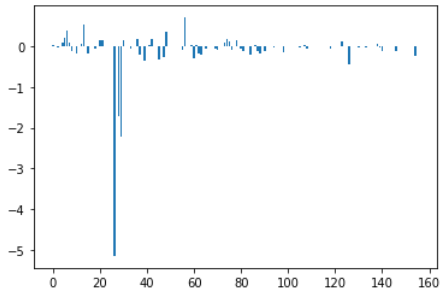
```
logreg2 = LogisticRegression(C=0.1, penalty='l1', solver='liblinear', random_state=42)
logreg2.fit(X_train,y_train)
y_pred2 = logreg2.predict(X_test)


print("Accuracy:", logreg2.score(X_test, y_test))
```

```
     Accuracy: 0.6286714713081316
```

Select best variables by looking at coefficients of variables and fit model with best variables and best parameters

```
# get importance
importance = logreg2.coef_[0]
vars = final_X.columns
count = 0
not_important = []
# summarize feature importance
for i,v in enumerate(importance):
    #print('Feature: %0d, Score: %.5f' % (i,v))
    if abs(v) < 0.000001:
      #print(vars[i], v)
      not_important.append(vars[i])
      count += 1
print("Count is", count)

# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
     Count is 84
```



From the above coefficients, we could make some feature selection and delete those with 0 coefficient.

```
selected_X1 = final_X.drop(columns=not_important)


X_train1, X_test1, y_train1, y_test1 = train_test_split(selected_X1, y, test_size=0.3, random_state=42)


logreg3 = LogisticRegression()
clf2 = GridSearchCV(logreg3,                     # model
                   param_grid = parameters,    # hyperparameters
                   scoring='accuracy',        # metric for scoring
                   cv=5)                     # number of folds
clf2.fit(X_train1, y_train1)

print("Tuned Hyperparameters :", clf2.best_params_)
print("Accuracy :", clf2.best_score_)
```

```
     Tuned Hyperparameters : {'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
     Accuracy : 0.6317438309183593
```

```
logreg_final = LogisticRegression(C=0.1, penalty='l1', solver='liblinear', random_state=42)
logreg_final.fit(X_train1, y_train1)
y_pred_final = logreg_final.predict(X_test1)
print("Accuracy:", logreg_final.score(X_test1, y_test1))
```

```
     Accuracy: 0.6287054664128365
```

Try LogisticRegressionCV

[ ] ↳ *2 cells hidden*

Model Result

For train data

```
my_results_train = logreg_final.fit(X_train1, y_train1)
predictions_train = my_results_train.predict(X_train1)
predictions_train_prob = my_results_train.predict_proba(X_train1)

print("The Classification table is", metrics.confusion_matrix(y_train1, predictions_train))
print(metrics.classification_report(y_train1, predictions_train))
```

```
The Classification table is [[28309  8407]
 [16766 15155]]
              precision    recall  f1-score   support

          NO       0.63      0.77      0.69     36716
         YES       0.64      0.47      0.55     31921

    accuracy                           0.63     68637
   macro avg       0.64      0.62      0.62     68637
weighted avg       0.64      0.63      0.62     68637
```

Find Proper Threshold

```
probs = np.linspace(0, 1, 11)
for i in range(10):
    y_pred = np.where(logreg_final.predict_proba(X_test1)[:,1] > probs[i], 'YES', 'NO')
    #print(y_pred)
    print('Threshold is', round(probs[i],2))
    accuracy = metrics.accuracy_score(y_test1, y_pred)
    print('Accuracy = ', accuracy)
```

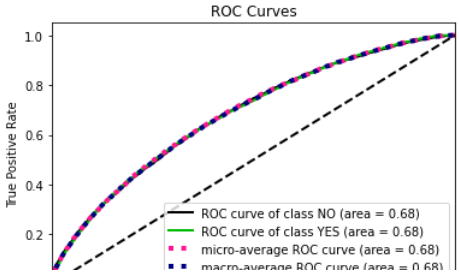
```
Threshold is 0.0
Accuracy =  0.46892847429970086
Threshold is 0.1
Accuracy =  0.48725183573565406
Threshold is 0.2
Accuracy =  0.492997008430786
Threshold is 0.3
Accuracy =  0.5290998096274137
Threshold is 0.4
Accuracy =  0.6038550448735382
Threshold is 0.5
Accuracy =  0.6287054664128365
Threshold is 0.6
Accuracy =  0.6016453630677182
Threshold is 0.7
Accuracy =  0.5744832744084851
Threshold is 0.8
Accuracy =  0.5546301332608105
Threshold is 0.9
Accuracy =  0.5413380473211857
```

From the above result, I would still choose the threshold to be 0.5.

For test data

```
my_results_train = logreg_final.fit(X_train1,y_train1)
predictions_test = my_results_train.predict(X_test1)
predictions_test_prob = my_results_train.predict_proba(X_test1)

print("The Classification table is", metrics.confusion_matrix(y_test1, predictions_test))
print(metrics.classification_report(y_test1, predictions_test))

fpr, tpr, tholds = metrics.roc_curve(y_test1, predictions_test_prob[:,1], pos_label = 1)
scplt.metrics.plot_roc(y_test1, predictions_test_prob)
plt.show()

#print("The AUC Value for the model is", metrics.auc(fpr, tpr))
```

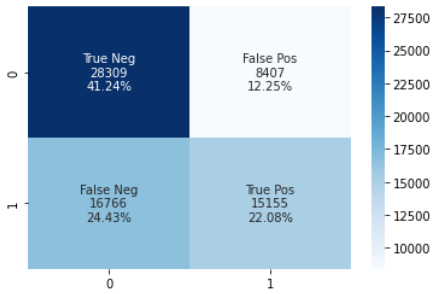
```
The Classification table is [[11966  3656]
 [ 7266  6528]]
              precision    recall  f1-score   support

          NO       0.62      0.77      0.69     15622
         YES       0.64      0.47      0.54     13794

    accuracy                           0.63     29416
   macro avg       0.63      0.62      0.62     29416
weighted avg       0.63      0.63      0.62     29416

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_ranking.py:1001: UndefinedMetricWarning: No positive samples in y_true, true positive value shou
  UndefinedMetricWarning,
```



Confusion Matrix

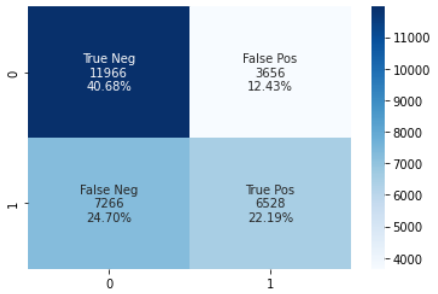
```
# For training data
cf_matrix_train = metrics.confusion_matrix(y_train1, predictions_train)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix_train.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix_train.flatten()/np.sum(cf_matrix_train)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix_train, annot=labels, fmt='', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f078547f990>
```




```
# For test data
cf_matrix = metrics.confusion_matrix(y_test1, predictions_test)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f07853bf610>
```



From the above classification table and confusion matrix, I would say this is a stable model since the proportion of 0s (No) and 1s (Yes)

Part 2. Classification Tree

- Perform GridSearchCv on DecisionTreeClassifier parameters (‘max_depth’, ‘criterion’, ‘max_features’, ‘max_leaf_nodes’ etc..)
- Use 5 - 10 fold cross validation

```
# GridSearch
parameters_refine = {
    'max_features': ['auto', 'sqrt', 'log2'],
    'criterion': ['entropy', 'gini'],
    'max_depth': [3, 4, 5, 8, 10],
    'splitter': ['best', 'random'],
    'min_samples_split': [2, 3, 5, 8],
    'min_samples_leaf': [1,5,8,10]
}
dt = GridSearchCV(DecisionTreeClassifier(), parameters_refine, cv=10)
dt.fit(X_train, y_train)
dt.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 'sqrt',
 'min_samples_leaf': 10,
 'min_samples_split': 2,
 'splitter': 'best'}
```

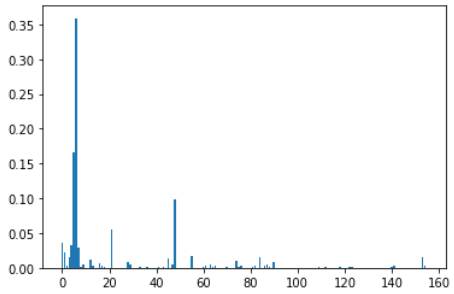
```
dt = DecisionTreeClassifier(criterion='gini', max_depth=8, max_features='sqrt', min_samples_leaf=10, min_samples_split=2, splitter='best', random_state=42)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_train)
accuracy = metrics.accuracy_score(y_train, y_pred)
print('Accuracy = ', accuracy)
print(dt.score(X_train, y_train))
```

```
Accuracy =  0.5934699943179335
0.5934699943179335
```

- Select best parameters and important features (10-20) from selected model

Feature Selection

```
# get importance
importance = dt.feature_importances_

# summarize feature importance
# for i,v in enumerate(importance):
#     temp1.append(X_train.columns[i], v)
    # if v == 0:
    #     print(X_train.columns[i], ':')
    #     print('Feature: %0d, Score: %.5f' % (i,v))

# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```




```
# get feature importance
temp1 = {}
for i,v in enumerate(importance):
    temp1[X_train.columns[i]] = v

import operator
sorted_d = dict(sorted(temp1.items(), key=operator.itemgetter(1),reverse=True))


count = 0
```

```
important_feature = []
for key in sorted_d.keys():
    print(key, ": ", sorted_d[key])
    important_feature.append(key)
    count += 1
    if count > 20:
        break
```

```
number_inpatient :  0.3589386648666915
number_emergency :  0.1657808384079481
adm_source_id_7 :  0.09897215130599091
disch_dispo_id_6 :  0.05472047092853081
time_in_hospital :  0.03630335105216611
number_outpatient :  0.03183313079073485
number_diagnoses :  0.029739257367675926
num_lab_procedures :  0.021735154722899167
adm_source_id_17 :  0.017233798764450078
num_medications :  0.015000760886194679
A1Cresult_Norm :  0.014741142241399163
change_No :  0.01471529836124385
adm_source_id_4 :  0.014050182347742503
adm_type_id_5 :  0.011355740781313288
age_[70-80) :  0.010521850961975691
repaglinide_No :  0.008094708447713267
disch_dispo_id_13 :  0.007614322064716271
disch_dispo_id_1 :  0.007372412502404585
disch_dispo_id_14 :  0.005090649344712939
adm_type_id_2 :  0.004948370567673284
adm_source_id_6 :  0.004754534675483632
```

Build model after feature selection

- Build a best model on train dataset

```
selected_X2 = final_X[important_feature]
X_train2, X_test2, y_train2, y_test2 = train_test_split(selected_X2, y, test_size=0.3, random_state=42)
```

```
dt_final = DecisionTreeClassifier(criterion='gini', max_depth=8, max_features='sqrt', min_samples_leaf=10, min_samples_split=2, splitter='best', random_state
dt_final.fit(X_train2, y_train2)
pred_train = dt_final.predict(X_train2)
accuracy = metrics.accuracy_score(y_train2, pred_train)
print('Accuracy = ', accuracy)
```

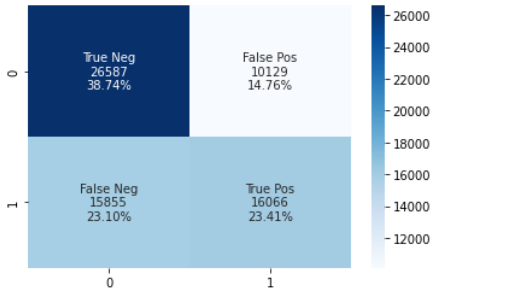
```
Accuracy =  0.6214286754957239
```

Confusion Matrix

- Generate confusion matrix on train

```
cf_matrix = metrics.confusion_matrix(y_train, pred_train)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f404502ed10>
```



- Plot tree using pydotplus , export_graphviz package
- Interpret and comment on the tree. Can you make sense of the terminal nodes? Point out significant interactions you think you see

Plot tree

[ ] ↳ 2 cells hidden

Make Prediction & Create Confusion matrix

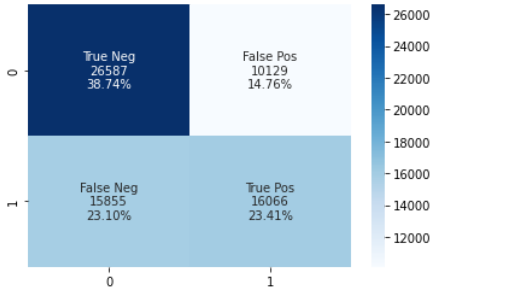
- Predict values for test data and compare confusion matrix with train. Are results stable?

```
pred_test = dt_final.predict(X_test2)
accuracy = metrics.accuracy_score(y_test2, pred_test)
print('Accuracy = ', accuracy)
```

```
Accuracy =  0.6137136252379657
```

```
cf_matrix_test = metrics.confusion_matrix(y_test, pred_test)
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4043a8d050>
```



Insight

↳ 2 cells hidden