TTDS: Machine Learning project

Accuracy Improved Random Forest Algorithm

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as plt
from matplotlib import pyplot
import matplotlib.pyplot as plt
```

In [2]:
```
data=pd.read_csv("D:/DataSets/diabetes.csv")
```

In [3]:
```
data
```

Out[3]:

	preg	glucose	bp_diastolic	skin_triceps	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

Dataset Extension

In [4]:
```
# Generate synthetic data by doubling the 'label' values
data_synthetic = data.copy()

# Concatenate the original and synthetic DataFrames
df = pd.concat([data, data_synthetic], ignore_index=True)

# Display the extended DataFrame
df
```

Out[4]:

	preg	glucose	bp_diastolic	skin_triceps	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
1531	10	101	76	48	180	32.9	0.171	63	0
1532	2	122	70	27	0	36.8	0.340	27	0
1533	5	121	72	23	112	26.2	0.245	30	0
1534	1	126	60	0	0	30.1	0.349	47	1
1535	1	93	70	31	0	30.4	0.315	23	0

1536 rows × 9 columns

In [5]:
```
df.head(10)
```

Out[5]:

	preg	glucose	bp_diastolic	skin_triceps	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

In [6]:
```
df.tail()
```

Out[6]:

	preg	glucose	bp_diastolic	skin_triceps	insulin	bmi	pedigree	age	label
1531	10	101	76	48	180	32.9	0.171	63	0
1532	2	122	70	27	0	36.8	0.340	27	0
1533	5	121	72	23	112	26.2	0.245	30	0
1534	1	126	60	0	0	30.1	0.349	47	1
1535	1	93	70	31	0	30.4	0.315	23	0

In [149]:
```
df.columns
```

Out[149]:
```
Index(['preg', 'glucose', 'bp_diastolic', 'skin_triceps', 'insulin', 'bmi',
       'pedigree', 'age', 'label'],
      dtype='object')
```

In [8]:
```
print("Number of Row in the Dataset:", df.shape[0])
print("Number of Columns in the Dataset:", df.shape[1])
```

```
Number of Row in the Dataset: 1536
Number of Columns in the Dataset: 9
```

Missing Values:

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
preg            0
glucose         0
bp_diastolic    0
skin_triceps    0
insulin         0
bmi             0
pedigree        0
age             0
label           0
dtype: int64
```

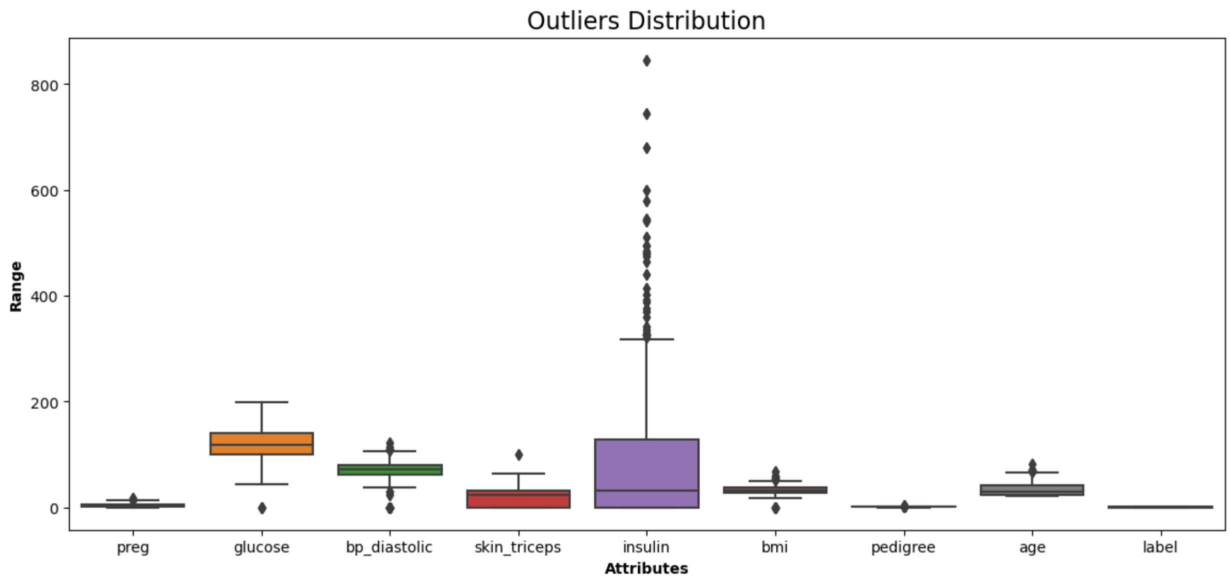
In [15]:
```
#check missing Values in the Dataset
missing_data=df.isnull()
for column in missing_data.columns.values.tolist():
    print(column)
    print(missing_data[column].value_counts())
    print("")
```

```
preg
False    1536
Name: preg, dtype: int64

glucose
False    1536
Name: glucose, dtype: int64

bp_diastolic
False    1536
Name: bp_diastolic, dtype: int64

skin_triceps
False    1536
Name: skin_triceps, dtype: int64

insulin
False    1536
Name: insulin, dtype: int64

bmi
False    1536
Name: bmi, dtype: int64

pedigree
False    1536
Name: pedigree, dtype: int64

age
False    1536
Name: age, dtype: int64

label
False    1536
Name: label, dtype: int64
```

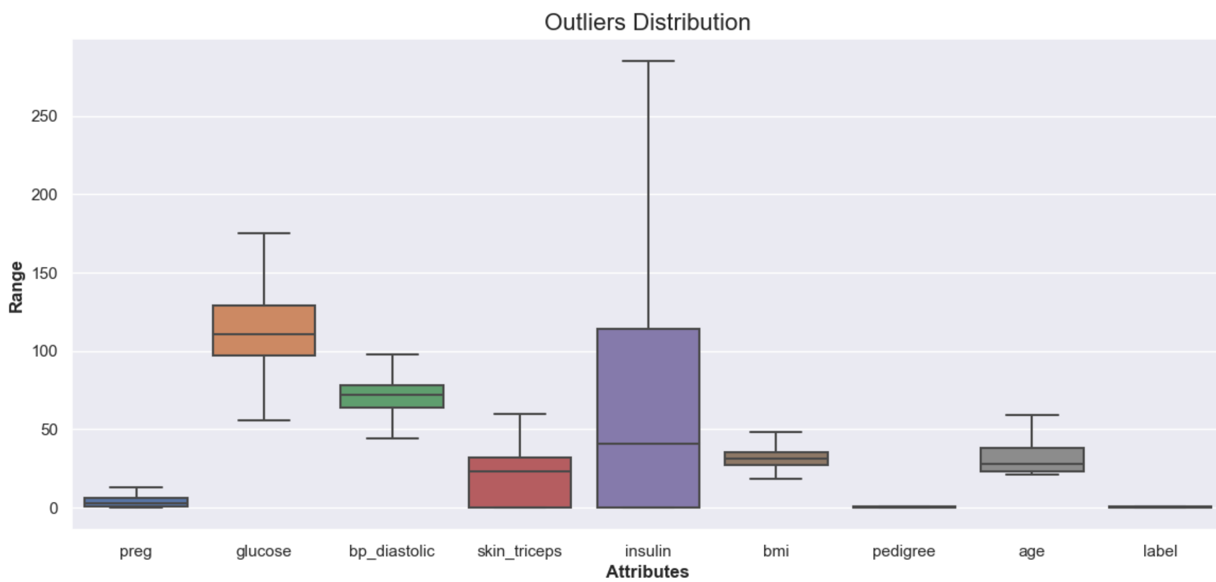
No missing values found in the dataset, therefore data doesn't need to be drop or replace.

Outliers Analysis

In [16]:
```
def show_boxplot(df):
    plt.rcParams['figure.figsize'] = [14,6]
    sns.boxplot(data = df, orient="v")
    plt.title("Outliers Distribution", fontsize = 16)
    plt.ylabel("Range", fontweight = 'bold')
    plt.xlabel("Attributes", fontweight = 'bold')
show_boxplot(df)
```



In [109]:
```
#Function 1st time
def remove_outliers(data):

    df = data.copy()

    for col in list(df.columns):
        Q1 = df[str(col)].quantile(0.25)
        Q3 = df[str(col)].quantile(0.75)
        IQR = Q3 - Q1
        # Define the lower and upper bounds to filter outliers
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        df = df[(df[str(col)] >= lower_bound) & (df[str(col)] <= upper_bound)]

    return df
without_outliers = remove_outliers(df)
show_boxplot(without_outliers)
```



NOTE: Outliers completely removed, after function has been run two to three times.

In [110]:
```
df=without_outliers
```

Data Formating

In [7]:
```
df.dtypes
```

Out[7]:
```
preg              int64
glucose           int64
bp_diastolic      int64
skin_triceps      int64
insulin           int64
bmi             float64
pedigree        float64
age               int64
label             int64
dtype: object
```

Exploratory Data Analysis (EDA)

Descriptive Satistics:

In [12]:
```
df.describe().T
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
preg	1536.0	3.845052	3.368480	0.000	1.00000	3.0000	6.00000	17.00
glucose	1536.0	120.894531	31.962202	0.000	99.00000	117.0000	140.25000	199.00
bp_diastolic	1536.0	69.105469	19.349501	0.000	62.00000	72.0000	80.00000	122.00
skin_triceps	1536.0	20.536458	15.947021	0.000	0.00000	23.0000	32.00000	99.00
insulin	1536.0	79.799479	115.206457	0.000	0.00000	30.5000	127.25000	846.00
bmi	1536.0	31.992578	7.881592	0.000	27.30000	32.0000	36.60000	67.10
pedigree	1536.0	0.471876	0.331221	0.078	0.24375	0.3725	0.62625	2.42
age	1536.0	33.240885	11.756400	21.000	24.00000	29.0000	41.00000	81.00
label	1536.0	0.348958	0.476796	0.000	0.00000	0.0000	1.00000	1.00

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1536 entries, 0 to 1535
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   preg          1536 non-null   int64  
 1   glucose       1536 non-null   int64  
 2   bp_diastolic  1536 non-null   int64  
 3   skin_triceps  1536 non-null   int64  
 4   insulin       1536 non-null   int64  
 5   bmi           1536 non-null   float64
 6   pedigree      1536 non-null   float64
 7   age           1536 non-null   int64  
 8   label         1536 non-null   int64  
dtypes: float64(2), int64(7)
memory usage: 108.1 KB
```

In [143]:
```
df.corr()
```

Out[143]:

	preg	glucose	bp_diastolic	skin_triceps	insulin	bmi	pedigree	age	label
preg	1.000000	0.135459	0.196070	-0.094739	-0.124026	0.035903	0.023478	0.639163	0.248129
glucose	0.135459	1.000000	0.221945	-0.005213	0.229386	0.152932	0.055723	0.219497	0.429107
bp_diastolic	0.196070	0.221945	1.000000	0.012043	-0.053348	0.251797	0.007455	0.330652	0.125304
skin_triceps	-0.094739	-0.005213	0.012043	1.000000	0.486622	0.368988	0.149839	-0.123725	0.018855
insulin	-0.124026	0.229386	-0.053348	0.486622	1.000000	0.179560	0.243634	-0.106722	0.070583
bmi	0.035903	0.152932	0.251797	0.368988	0.179560	1.000000	0.121557	0.084640	0.242767
pedigree	0.023478	0.055723	0.007455	0.149839	0.243634	0.121557	1.000000	0.021540	0.175675
age	0.639163	0.219497	0.330652	-0.123725	-0.106722	0.084640	0.021540	1.000000	0.289717
label	0.248129	0.429107	0.125304	0.018855	0.070583	0.242767	0.175675	0.289717	1.000000

In [148]:
```
#Total size of the Rows of dataset:
print("Total Rows of the dataset:", df.shape[0])

#Total size of the Columns of dataset:
print("Total Columns of the dataset:", df.shape[1])
```

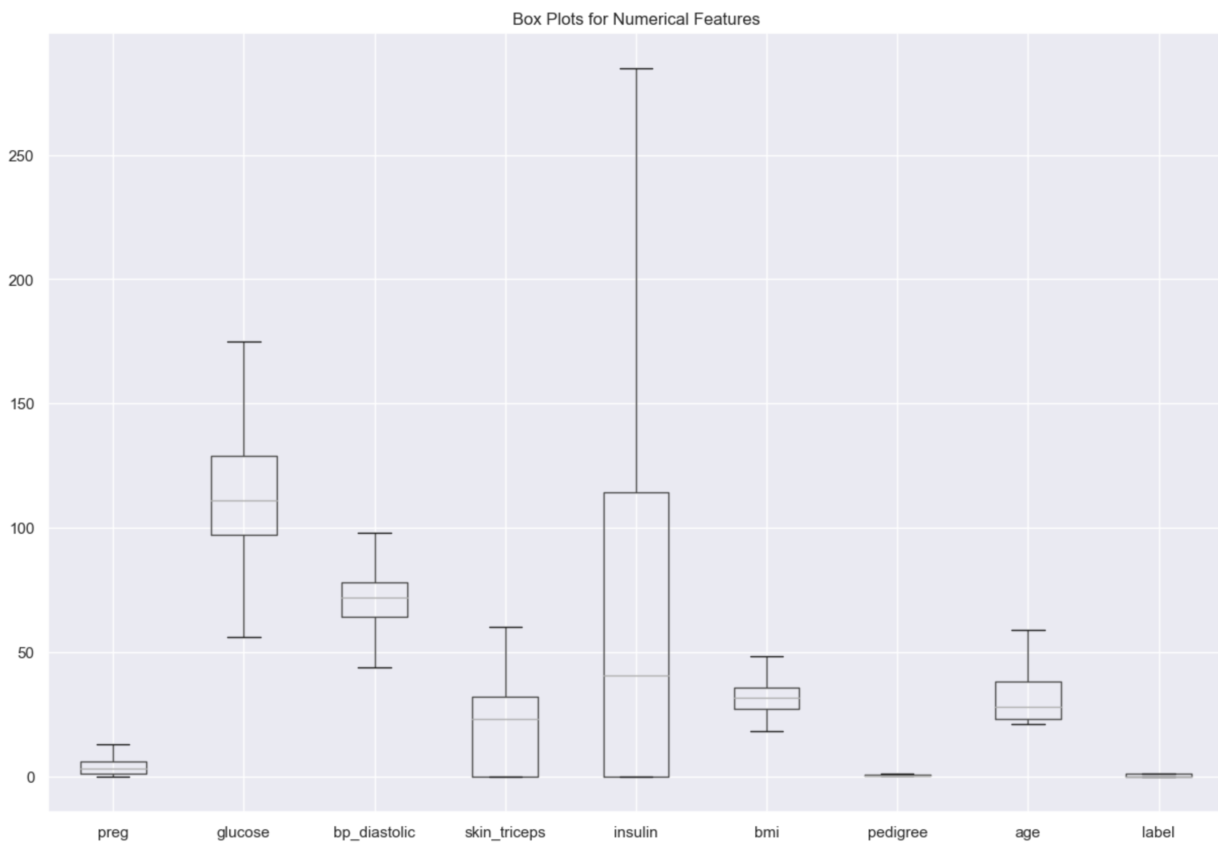
```
Total Rows of the dataset: 1112
Total Columns of the dataset: 9
```

In [144]:
```
#check label value count
df.label.value_counts()
```

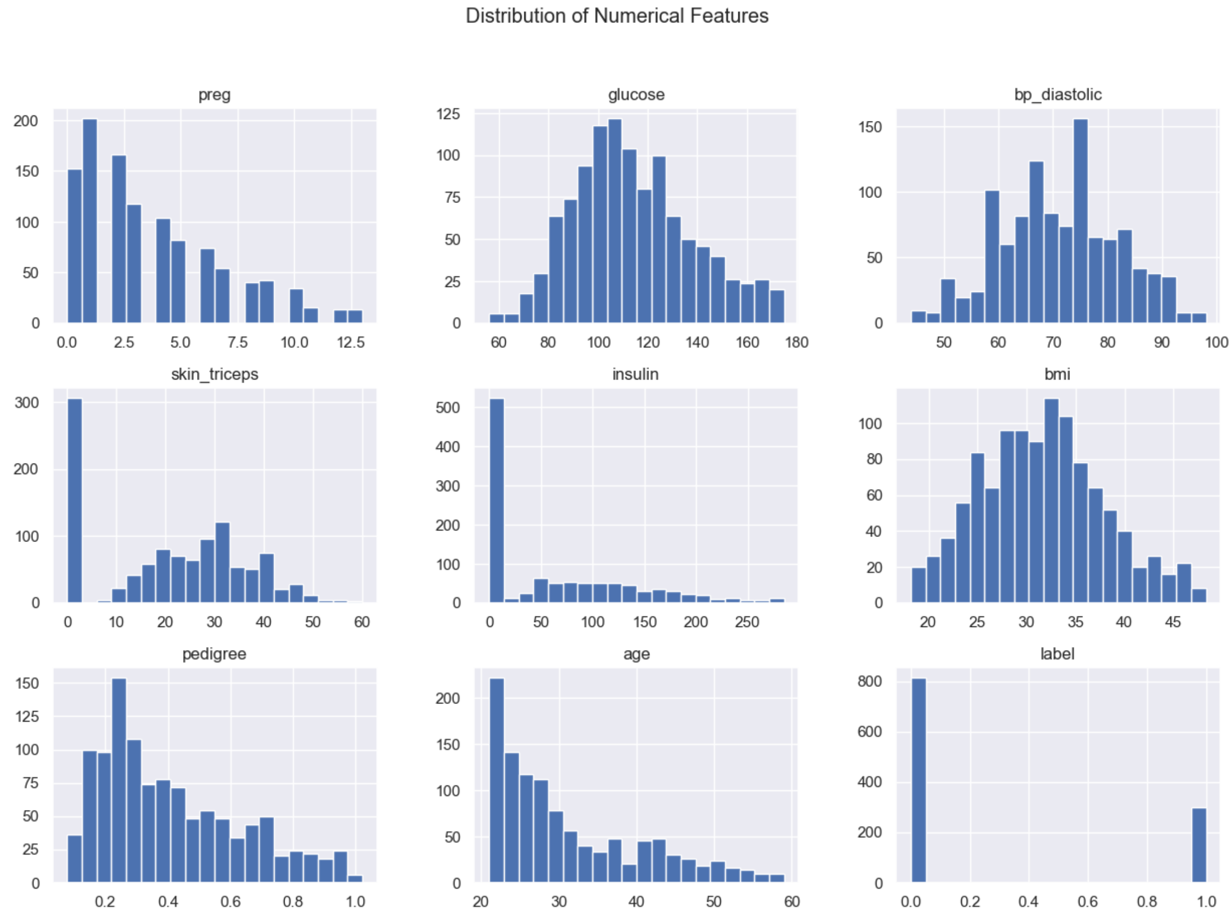
Out[144]:
```
0    814
1    298
Name: label, dtype: int64
```

Graphical Analysis

In [113]:
```
# Box plots for numerical features
plt.figure(figsize=(15, 10))
df.boxplot()
plt.title('Box Plots for Numerical Features')
plt.show()
```

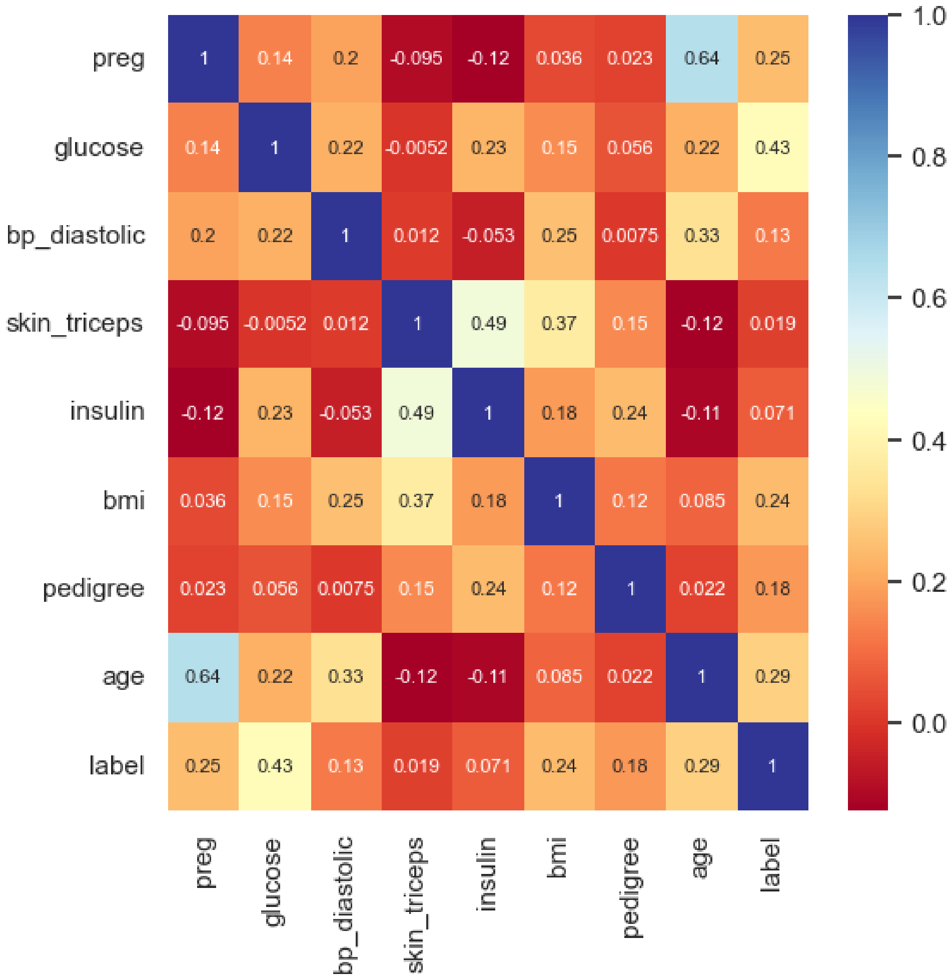


In [111]:
```
# Distribution of numerical features
df.hist(bins=20, figsize=(15, 10))
plt.suptitle('Distribution of Numerical Features')
plt.show()
```

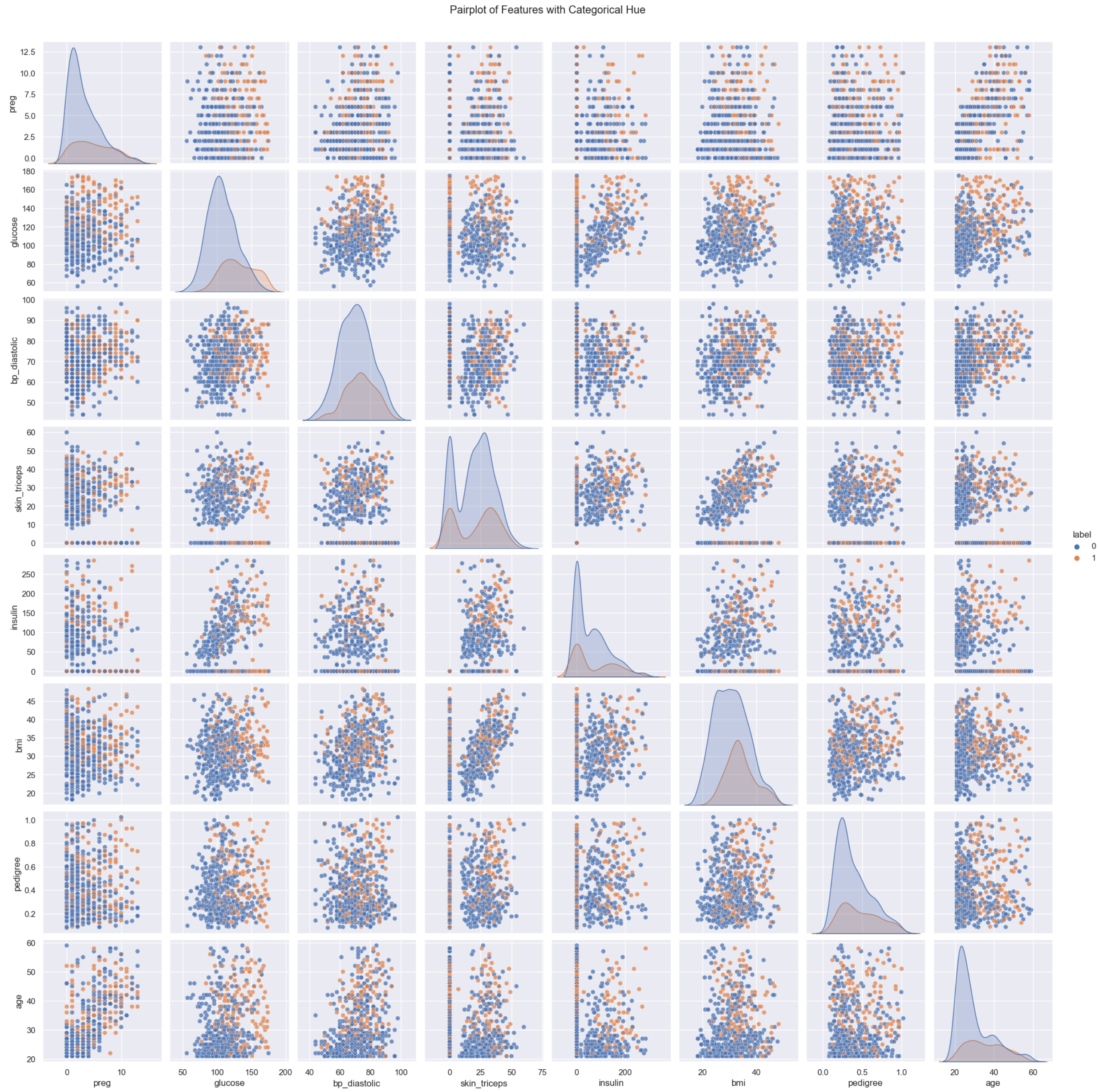


HeatMap

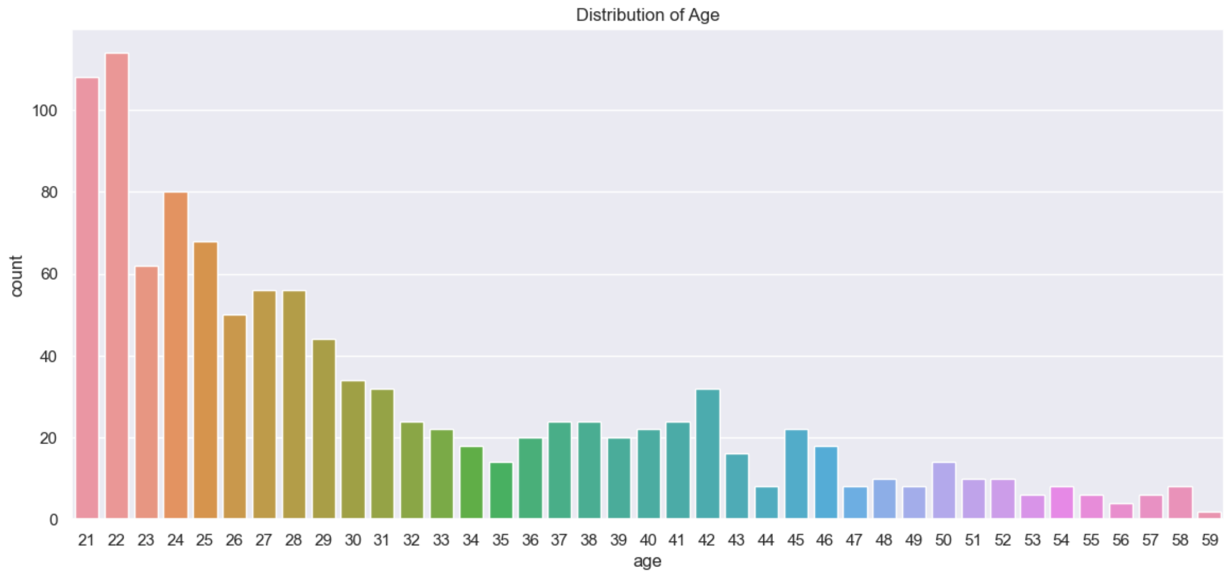
In [112]:
```
#get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(6,6))
#plot heat map
sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlBu", annot_kws={"fontsize": 8});
```



In [147]:
```
# Pairplot with categorical hue for better insight into relationships
sns.pairplot(df, hue='label', diag_kind='kde', plot_kws={'alpha': 0.5})
plt.suptitle('Pairplot of Features with Categorical Hue', y=1.02)
plt.show()
```



In [141]:
```
# Distribution of categorical features
sns.countplot(x='age', data=df)
plt.title('Distribution of Age')
plt.show()
```



In [118]:
```
cols=list(df.columns)
cols
```

Out[118]:
```
['preg',
 'glucose',
 'bp_diastolic',
 'skin_triceps',
 'insulin',
 'bmi',
 'pedigree',
 'age',
 'label']
```

In [121]:
```
feature_cols=['preg', 'glucose', 'bp_diastolic', 'skin_triceps', 'insulin', 'bmi', 'pedigree',
print(feature_cols)
```

```
['preg', 'glucose', 'bp_diastolic', 'skin_triceps', 'insulin', 'bmi', 'pedigree', 'age']
```

Data Train-Test split

In [122]:
```
#Library Call for data split in two portion Train and Test:
from sklearn.model_selection import train_test_split
```

In [123]:
```
#dataframe
x=df[feature_cols]  #feature
#series
y=df.label
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =0.25, random_state=30)
```

In [124]:
```
#Total size of the Training dataset:
print("[XY_Train] dataset Shape:", x_train.shape)

#Total size of the Testing dataset:
print("[XY_Test] dataset Shape:", x_test.shape)
```

```
[XY_Train] dataset Shape: (834, 8)
[XY_Test] dataset Shape: (278, 8)
```

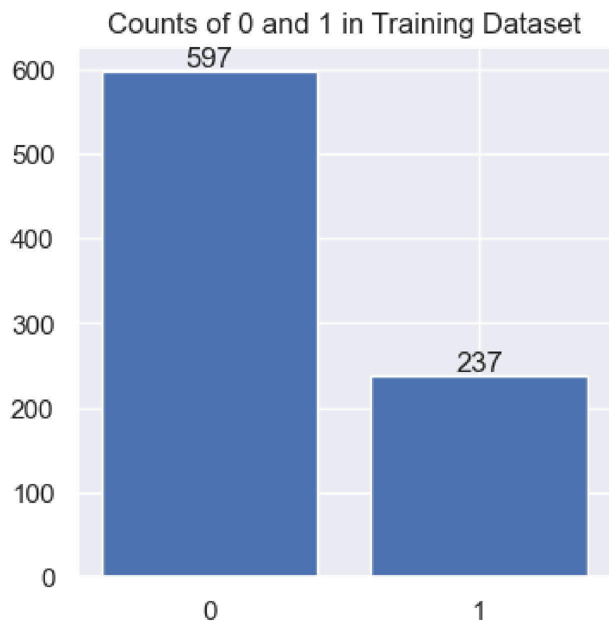
In [127]:
```
#Checking the number of 0's in Training portion of the Dataset:
print("[Y_Train] Total number of [0] in dataset :", len(y_train[y_train==0]))

#Checking the number of 1's in Training portion of the Dataset:
print("[Y_Train] Total number of [1] in dataset :", len(y_train[y_train==1]))
```

```
[Y_Train] Total number of [0] in dataset : 597
[Y_Train] Total number of [1] in dataset : 237
```

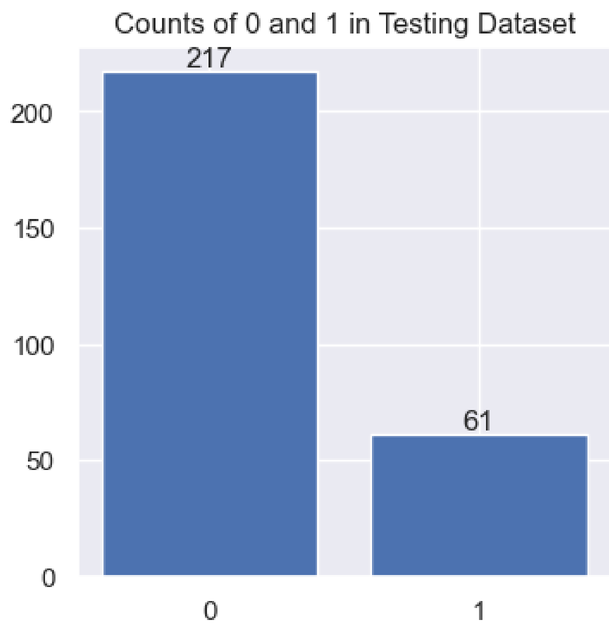
In [128]:
```
#Checking the number of 0's in Testing portion of the Dataset:
print("[Y_Test] Total number of [0] in dataset :", len(y_test[y_test==0]))

#Checking the number of 1's in Testing portion of the Dataset:
print("[Y_Test] Total number of [1] in dataset :", len(y_test[y_test==1]))
```

```
[Y_Test] Total number of [0] in dataset : 217
[Y_Test] Total number of [1] in dataset : 61
```

In [129]:
```
# get total number of 0 in the training dataset
Trcount0 = y_train[y_train==0].count()

# get total number of 1 in the training dataset
Trcount1 = y_train[y_train==1].count()

# Plotting the bar chart
label = ['0', '1']
counts = [Trcount0, Trcount1]

plt.figure(figsize=(4,4))
plt.title('Counts of 0 and 1 in Training Dataset')
plt.bar(label, counts)

# Add annotations to the bars
for i, count in enumerate(counts):
    plt.text(i, count, str(count), ha='center', va='bottom')

plt.show()
```



In [130]:
```
# get total number of 0 in the testing dataset
Trcount0 = y_test[y_test==0].count()

# get total number of 1 in the testing dataset
Trcount1 = y_test[y_test==1].count()

# Plotting the bar chart
label = ['0', '1']
counts = [Trcount0, Trcount1]

plt.figure(figsize=(4,4))
plt.title('Counts of 0 and 1 in Testing Dataset')
plt.bar(label, counts)

# Add annotations to the bars
for i, count in enumerate(counts):
    plt.text(i, count, str(count), ha='center', va='bottom')

plt.show()
```



Random Forest Lib Call

In [131]:
```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=3)
```

In [132]:
```
# Train Classifer
model = clf.fit(x_train, y_train)
```

Model

In [133]:
```
#Predict the response for test dataset
y_pred = clf.predict(x_test)
```

In [134]:
```
y=pd.DataFrame({"Origional": y_test, "Predicted": y_pred})
y.head()
```

Out[134]:

	Origional	Predicted
1458	0	0
280	1	0
318	0	0
850	0	0
1520	0	0

In [135]:
```
y.sample(10)
```

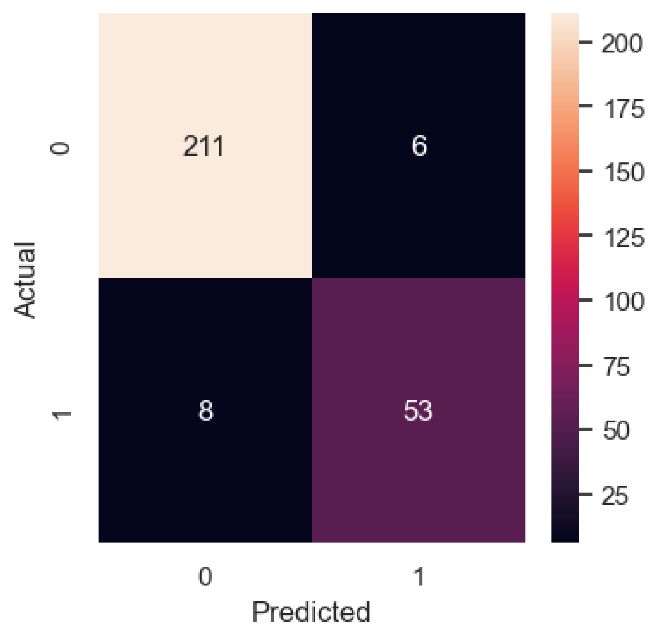
Out[135]:

	Origional	Predicted
714	0	0
1405	0	0
28	0	0
778	0	0
1417	0	0
881	0	0
720	0	0
334	0	0
1330	0	0
1199	0	0

Confusion Matrics

In [136]:
```
# calculate accuracy
from sklearn import metrics

result = metrics.confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)

def plt1():
    import seaborn as sns; sns.set()
    plt.figure(figsize=(4,4))
    c_mtrx = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
    sns.heatmap(c_mtrx, annot=True, fmt = '.3g')

plt1()
```

```
Confusion Matrix:
[[211   6]
 [  8  53]]
```



Accuracy Calculation

In [137]:
```
#[row, column]
#(Actual, Predict)
TP = result[1, 1]
TN = result[0, 0]
FP = result[0, 1]
FN = result[1, 0]
```

In [138]:
```
def EvClsMdl(res):
    print('Metrics computed from a confusion matrix')
    print("Accuracy:\t", metrics.accuracy_score(y_test, y_pred))
    print("Sensitivity:\t", metrics.recall_score(y_test, y_pred))
    print("Specificity:\t",TN / (TN + FP))
    print("Precision:\t", metrics.precision_score(y_test, y_pred))
    print("Classification Eerror:", 1 - metrics.accuracy_score(y_test, y_pred))
    print("False_Positive_Rate:", 1 - TN / (TN + FP))
    print('###########################################')
EvClsMdl(result)
```

```
Metrics computed from a confusion matrix
Accuracy:        0.9496402877697842
Sensitivity:     0.8688524590163934
Specificity:     0.9723502304147466
Precision:       0.8983050847457628
Classification Eerror: 0.05035971223021585
False_Positive_Rate: 0.027649769585253448
###########################################
```

KNN Algorithm

Before Improvement

```
Metrics computed from a confusion matrix
Accuracy:    0.734375
Sensitivity:     0.6521739130434783
Specificity:     0.7804878048780488
Precision:   0.625
Classification Eerror: 0.265625
False_Positive_Rate: 0.2195121951219512
###########################################
```

After Improvement

```
Metrics computed from a confusion matrix
Accuracy:    0.7955555555555556
Sensitivity:     0.6415094339622641
Specificity:     0.8430232558139535
Precision:   0.5573770491803278
Classification Eerror: 0.20444444444444443
False_Positive_Rate: 0.15697674418604646
###########################################
```

Decision Tree Algorithm

Before Improvement

```
Metrics computed from a confusion matrix
Accuracy:    0.734375
Sensitivity:     0.6521739130434783
Specificity:     0.7804878048780488
Precision:   0.625
Classification Eerror: 0.265625
False_Positive_Rate: 0.2195121951219512
```

```
############################################
```

After Improvement

```
Metrics computed from a confusion matrix
Accuracy:    0.84
Sensitivity:     0.3584905660377358
Specificity:     0.9883720930232558
Precision:   0.9047619047619048
Classification Eerror: 0.16000000000000003
False_Positive_Rate: 0.011627906976744207
############################################
```

Random Forest Accuracy

Before Improvement

```
Metrics computed from a confusion matrix
Accuracy:    0.8723958333333334
Sensitivity:     0.8043478260869565
Specificity:     0.9105691056910569
Precision:   0.8345864661654135
Classification Eerror: 0.12760416666666663
False_Positive_Rate: 0.08943089430894313
############################################
```

After Improvement

```
Metrics computed from a confusion matrix
Accuracy:    0.9496402877697842
Sensitivity:     0.8688524590163934
Specificity:     0.9723502304147466
Precision:   0.8983050847457628
Classification Eerror: 0.05035971223021585
False_Positive_Rate: 0.027649769585253448
############################################
```

In [ ]: