Heart disease remains a significant global health concern, causing a substantial number of deaths every year. Early and accurate diagnosis of heart disease can greatly improve patient outcomes and reduce mortality rates. In recent years, advancements in machine learning techniques have provided promising avenues for enhancing diagnostic accuracy by leveraging various patient data attributes.

This project aims to develop a predictive model for heart disease using machine learning algorithms. By analyzing a comprehensive set of patient data, including medical history, physiological parameters, and lifestyle factors, we seek to create a model that can effectively predict the presence or absence of heart disease in individuals. The utilization of machine learning in this context is particularly exciting due to its potential to uncover complex patterns and interactions within the data that might not be immediately apparent through traditional medical approaches. ¶

In this project, we will explore the entire machine learning pipeline, starting from data collection and preprocessing, feature selection, model training, hyperparameter tuning, and ultimately, evaluation of the model's performance. By undertaking this comprehensive analysis, we aim to not only build an accurate predictive model but also gain insights into the key factors that contribute to heart disease.

The dataset employed in this project comprises a diverse range of attributes, including demographic information, clinical measurements, and lifestyle choices. Through meticulous preprocessing and feature engineering, we aim to enhance the quality of the data and optimize its relevance for model training. Subsequently, a variety of machine learning algorithms will be employed, including but not limited to logistic regression, decision trees, random forests, and support vector machines. The models will be rigorously evaluated using appropriate metrics, such as accuracy, precision, recall, and F1-score, to gauge their predictive capabilities.

The outcomes of this project could have significant implications for both the medical and machine learning communities. Successful development of an accurate predictive model could aid healthcare professionals in making informed decisions about patient care. Additionally, it could showcase the potential of machine learning techniques in contributing to medical research and improving disease diagnosis.

In the following sections, we will delve into the specifics of data preprocessing, model development, and evaluation, providing a comprehensive overview of the entire project's methodology and outcomes.

In [ ]:

In [1]:
```
import pandas as pd
import numpy as np


%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree


from sklearn.model_selection import train_test_split
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
```

In [2]:
```
df = pd.read_csv('heart_dieases.csv')
df
```

Out[2]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	Di
0	No	16.60	Yes	No	No	3.0	30.0	
1	No	20.34	No	No	Yes	0.0	0.0	
2	No	26.58	Yes	No	No	20.0	30.0	
3	No	24.21	No	No	No	0.0	0.0	
4	No	23.71	No	No	No	28.0	0.0	
...	...	...	...	...	...	...	...	...
319790	Yes	27.41	Yes	No	No	7.0	0.0	
319791	No	29.84	Yes	No	No	0.0	0.0	
319792	No	24.24	No	No	No	0.0	0.0	
319793	No	32.81	No	No	No	0.0	0.0	
319794	No	46.56	No	No	No	0.0	0.0	

319795 rows × 18 columns

In [3]:
```
df.describe()
```

Out[3]:

	BMI	PhysicalHealth	MentalHealth	SleepTime
count	319795.000000	319795.00000	319795.000000	319795.000000
mean	28.325399	3.37171	3.898366	7.097075
std	6.356100	7.95085	7.955235	1.436007
min	12.020000	0.00000	0.000000	1.000000
25%	24.030000	0.00000	0.000000	6.000000
50%	27.340000	0.00000	0.000000	7.000000
75%	31.420000	2.00000	3.000000	8.000000
max	94.850000	30.00000	30.000000	24.000000

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319795 entries, 0 to 319794
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   HeartDisease      319795 non-null  object 
 1   BMI               319795 non-null  float64
 2   Smoking           319795 non-null  object 
 3   AlcoholDrinking   319795 non-null  object 
 4   Stroke            319795 non-null  object 
 5   PhysicalHealth    319795 non-null  float64
 6   MentalHealth      319795 non-null  float64
 7   DiffWalking       319795 non-null  object 
 8   Sex               319795 non-null  object 
 9   AgeCategory       319795 non-null  object 
 10  Race              319795 non-null  object 
 11  Diabetic          319795 non-null  object 
 12  PhysicalActivity  319795 non-null  object 
 13  GenHealth         319795 non-null  object 
 14  SleepTime         319795 non-null  float64
 15  Asthma            319795 non-null  object 
 16  KidneyDisease     319795 non-null  object 
 17  SkinCancer        319795 non-null  object 
dtypes: float64(4), object(14)
memory usage: 43.9+ MB
```

The output shows that we have around 319795 entries with 18 columns. No null values, and we have 14 numeric features, and 4 categorial features. We can convert the string attributes that has only two possibilities of unique values, but first let's make sure that there is no abnormal values

In [5]:
```
df.nunique()
```

Out[5]:
```
HeartDisease           2
BMI                 3604
Smoking                2
AlcoholDrinking        2
Stroke                 2
PhysicalHealth        31
MentalHealth          31
DiffWalking            2
Sex                    2
AgeCategory           13
Race                   6
Diabetic               4
PhysicalActivity       2
GenHealth              5
SleepTime             24
Asthma                 2
KidneyDisease          2
SkinCancer             2
dtype: int64
```

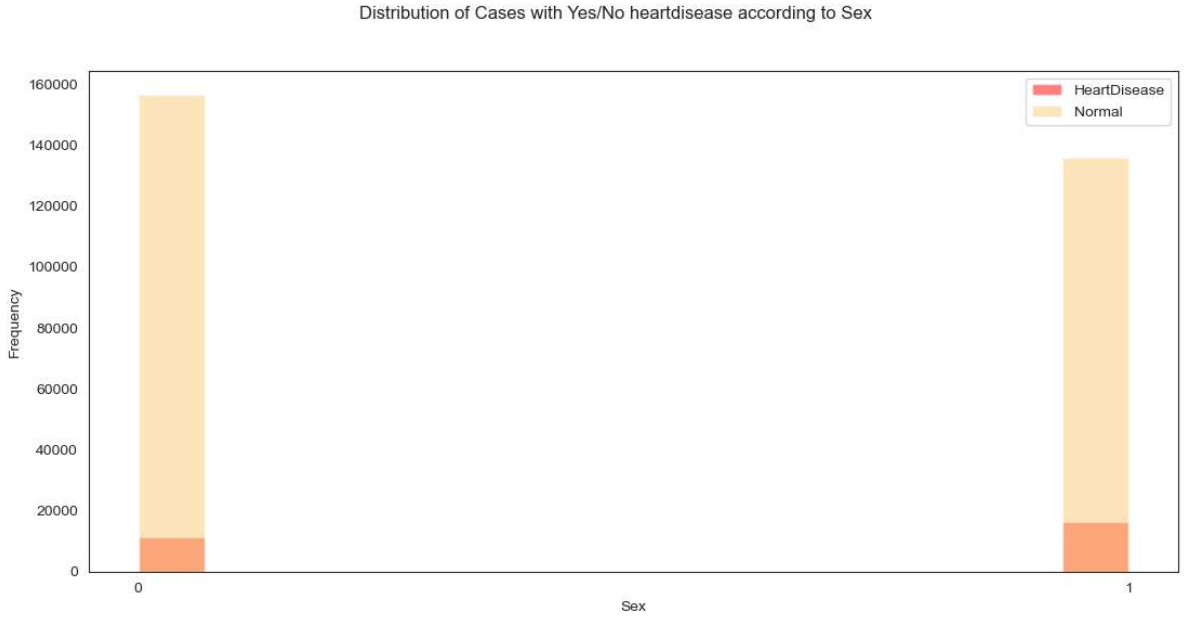
In [6]:
```
df =  df[df.columns].replace({'Yes':1, 'No':0, 'Male':1,'Female':0,'No, border
df['Diabetic'] = df['Diabetic'].astype(int)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Race                0
Diabetic            0
PhysicalActivity    0
GenHealth           0
SleepTime           0
Asthma              0
KidneyDisease       0
SkinCancer          0
dtype: int64
```

EXPLORATORY DATA ANALYSIS

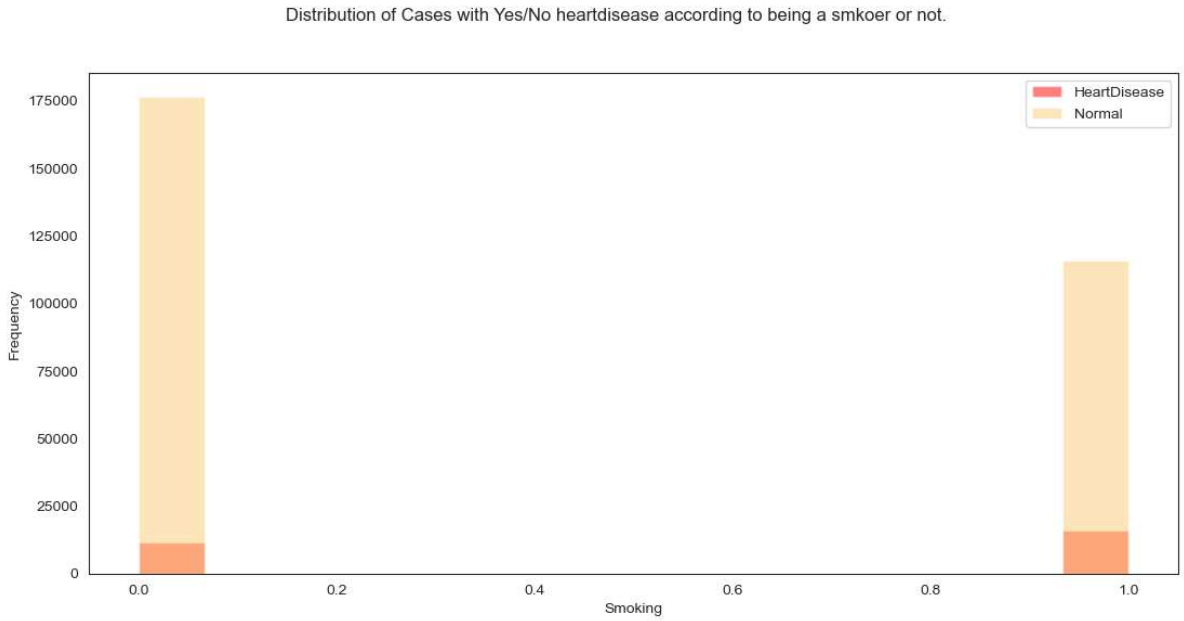
In [79]:
```
fig, ax = plt.subplots(figsize = (13,6))

ax.hist(df[df["HeartDisease"]==1]["Sex"], bins=15, alpha=0.5, color="red", lab
ax.hist(df[df["HeartDisease"]==0]["Sex"], bins=15, alpha=0.5, color="#fccc79",

ax.set_xlabel("Sex")
ax.set_ylabel("Frequency")
ax.set_xticks([0, 1])

fig.suptitle("Distribution of Cases with Yes/No heartdisease according to Sex"

ax.legend();
```



Most cases of heart disease are men .

Most cases that got no hart disease are women .

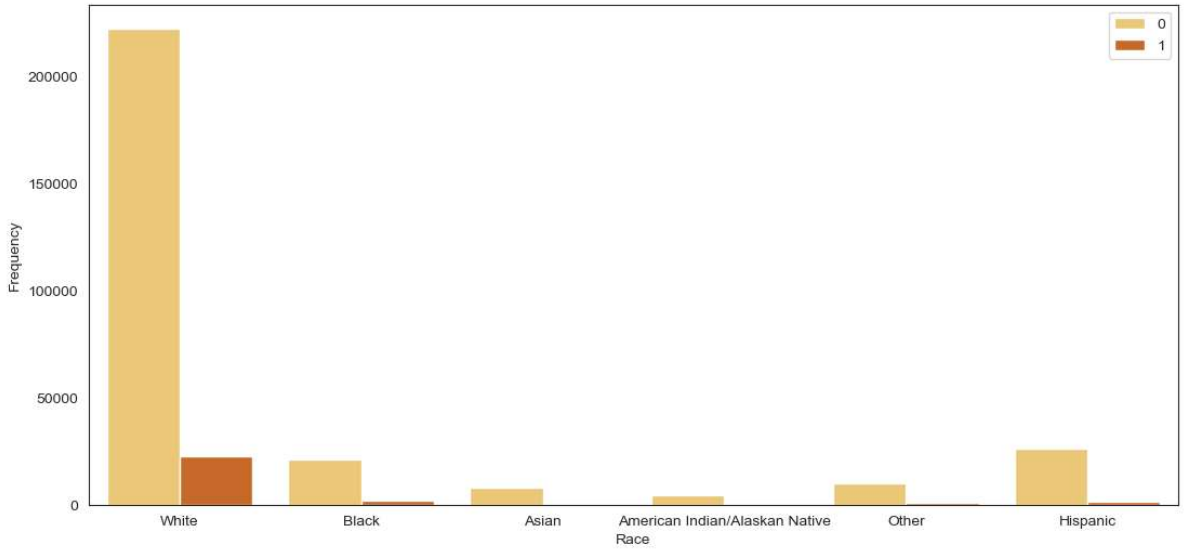
In [80]:
```
fig, ax = plt.subplots(figsize = (13,6))

ax.hist(df[df["HeartDisease"]==1]["Smoking"], bins=15, alpha=0.5, color="red",
ax.hist(df[df["HeartDisease"]==0]["Smoking"], bins=15, alpha=0.5, color="#fccc

ax.set_xlabel("Smoking")
ax.set_ylabel("Frequency")

fig.suptitle("Distribution of Cases with Yes/No heartdisease according to bein

ax.legend();
```



For smokers, they are the largest group that suffers from heart disease.

There are also cases of heart disease, although they don't smoke, this is related to other factors.

In [81]:
```
plt.figure(figsize = (13,6))
sns.countplot( x= df['Race'], hue = 'HeartDisease', data = df, palette = 'YlOr
plt.xlabel('Race')
plt.legend()
plt.ylabel('Frequency')
plt.show()
```

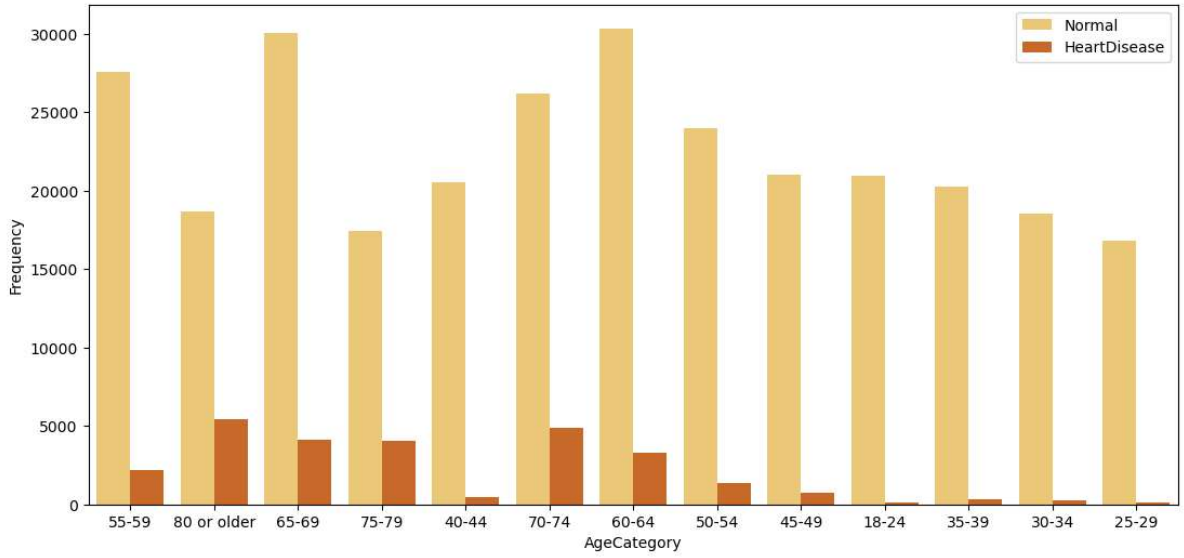


1 refers to heartdisease.

0 refers to normal state.

We can see that white people are more susceptible to heart disease.

In [16]:
```
plt.figure(figsize = (13,6))
sns.countplot(x = df['AgeCategory'], hue = 'HeartDisease', data = df, palette
fig.suptitle("Distribution of Cases with Yes/No hartdisease according to AgeCa
plt.xlabel('AgeCategory')
plt.legend(['Normal','HeartDisease'])
plt.ylabel('Frequency')
plt.show()
```



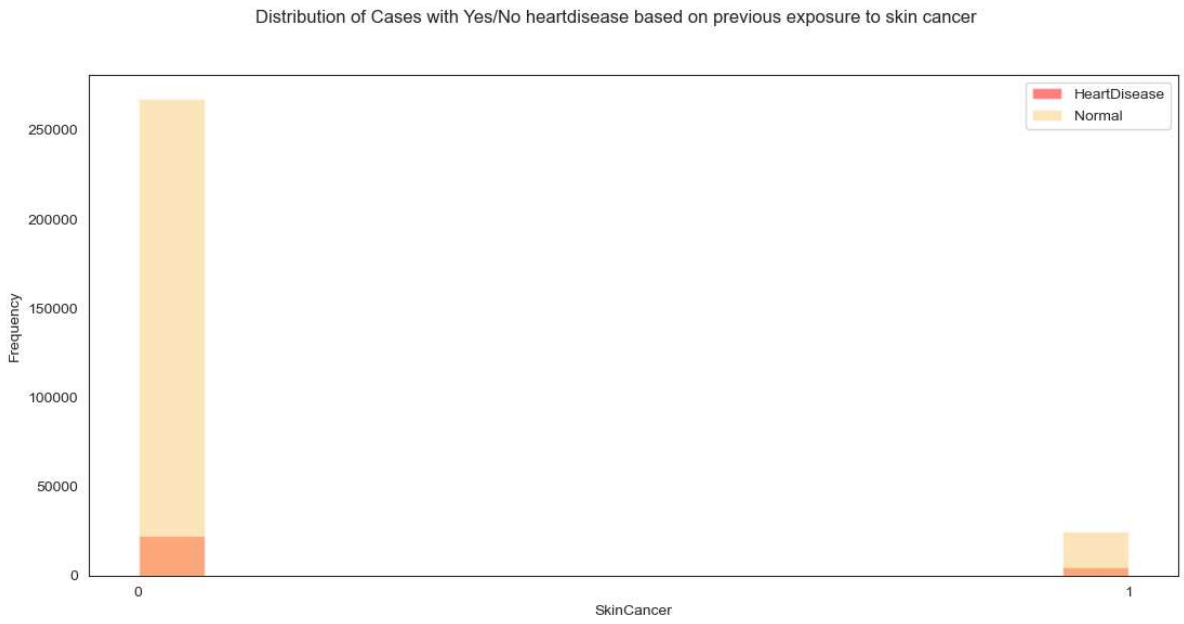
We can see that people who are 80 or older are more likely to get HeartDiseases.

In [82]:
```
fig, ax = plt.subplots(figsize = (13,6))

ax.hist(df[df["HeartDisease"]==1]["KidneyDisease"], bins=15, alpha=0.5, color=
ax.hist(df[df["HeartDisease"]==0]["KidneyDisease"], bins=15, alpha=0.5, color=

ax.set_xlabel("KidneyDisease")
ax.set_ylabel("Frequency")
ax.set_xticks([0, 1])


fig.suptitle("Distribution of Cases with Yes/No heartdisease according to kidn

ax.legend();
```

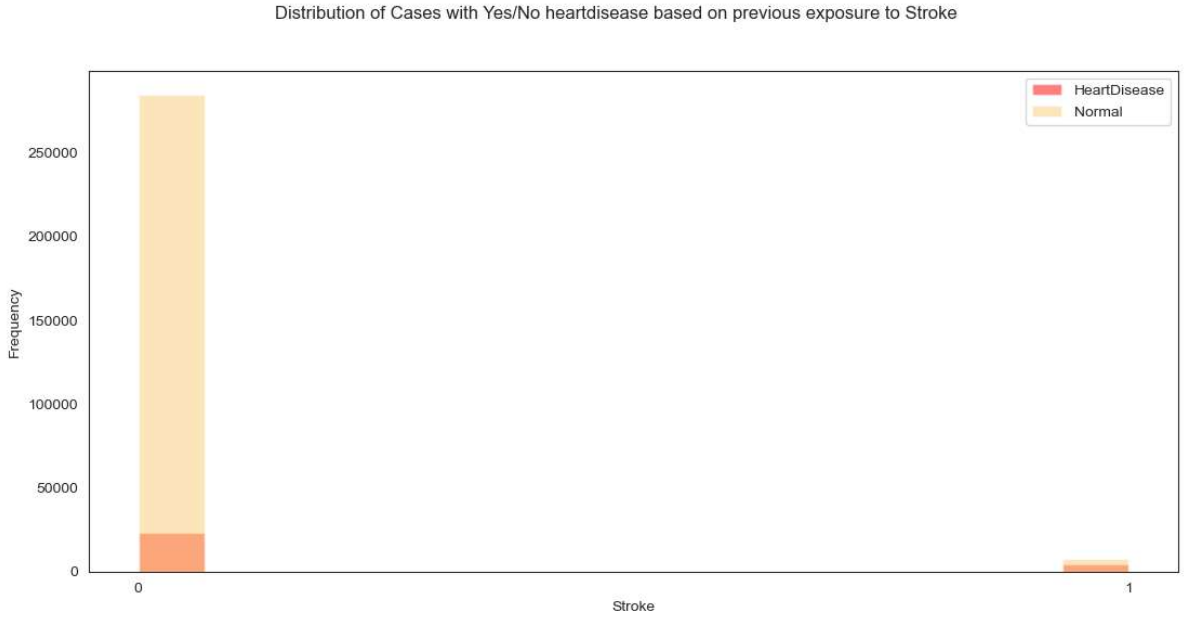
We can see that very less kidney disease and the number of non kideny disease having heart disease are more than that of kidney disease.

In [84]:
```
fig, ax = plt.subplots(figsize = (13,6))

ax.hist(df[df["HeartDisease"]==1]["SkinCancer"], bins=15, alpha=0.5, color="re
ax.hist(df[df["HeartDisease"]==0]["SkinCancer"], bins=15, alpha=0.5, color="#f

ax.set_xlabel("SkinCancer")
ax.set_ylabel("Frequency")
ax.set_xticks([0, 1])


fig.suptitle("Distribution of Cases with Yes/No heartdisease based on previous

ax.legend()
```

Out[84]:
```
<matplotlib.legend.Legend at 0x1e6379f3a00>
```

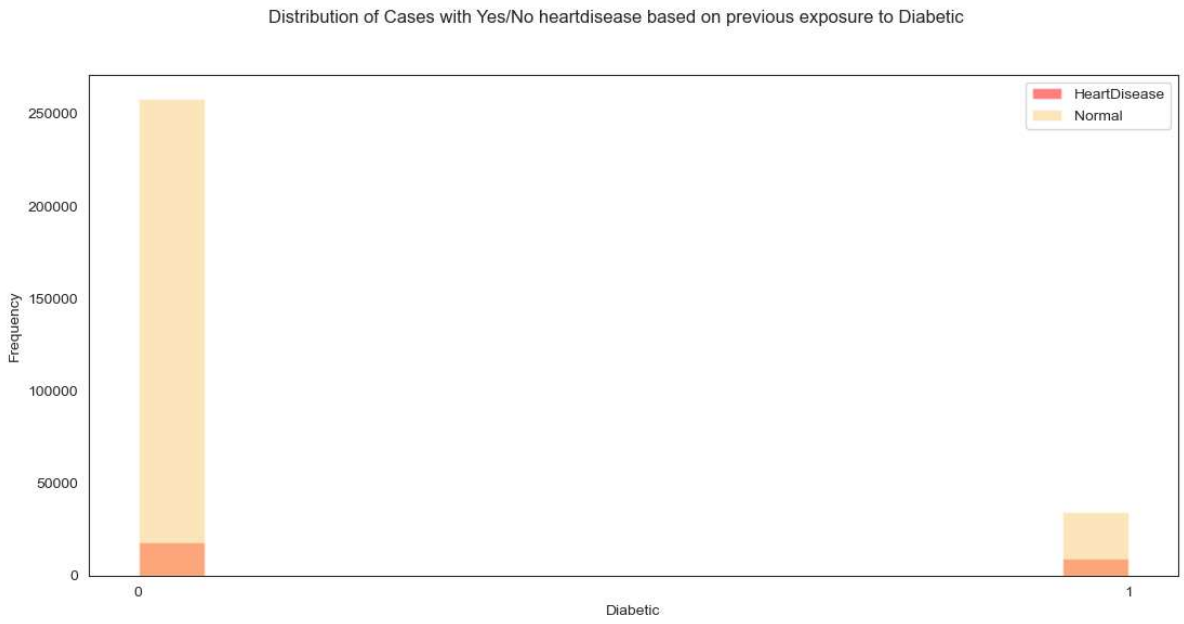


In [86]:
```
fig, ax = plt.subplots(figsize = (13,6))

ax.hist(df[df["HeartDisease"]==1]["Stroke"], bins=15, alpha=0.5, color="red",
ax.hist(df[df["HeartDisease"]==0]["Stroke"], bins=15, alpha=0.5, color="#fccc7

ax.set_xlabel("Stroke")
ax.set_ylabel("Frequency")
ax.set_xticks([0, 1])

fig.suptitle("Distribution of Cases with Yes/No heartdisease based on previous

ax.legend()
```



In [88]:
```
fig, ax = plt.subplots(figsize = (13,6))

ax.hist(df[df["HeartDisease"]==1]["Diabetic"], bins=15, alpha=0.5, color="red"
ax.hist(df[df["HeartDisease"]==0]["Diabetic"], bins=15, alpha=0.5, color="#fcc

ax.set_xlabel("Diabetic")
ax.set_ylabel("Frequency")
ax.set_xticks([0, 1])

fig.suptitle("Distribution of Cases with Yes/No heartdisease based on previous

ax.legend()
```

Out[88]: <matplotlib.legend.Legend at 0x1e628e82530>

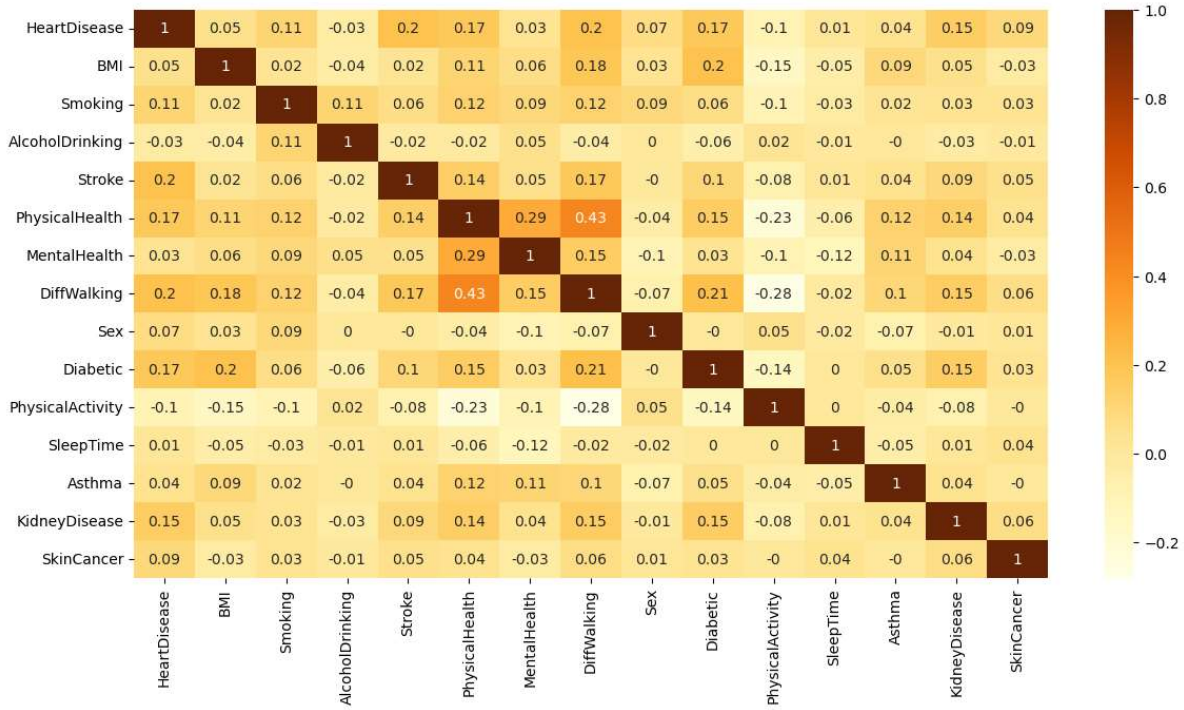


Visualization Of Numerical Features

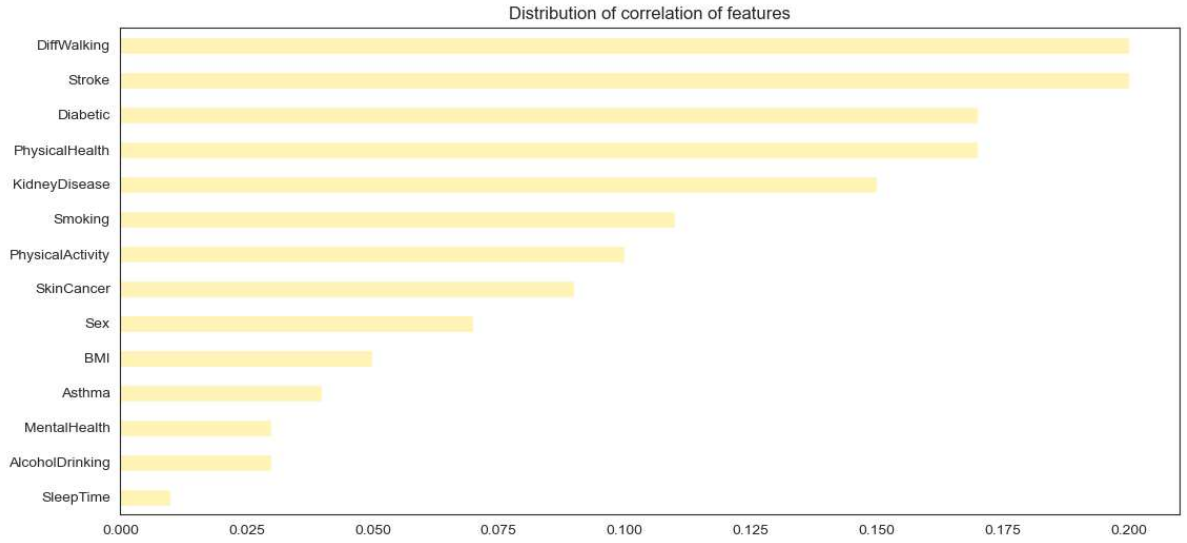
In [21]:
```
correlation = df.corr().round(2)
plt.figure(figsize = (14,7))
sns.heatmap(correlation, annot = True, cmap = 'YlOrBr')
```

```
C:\Users\Asus\AppData\Local\Temp\ipykernel_20068\1001107408.py:1: FutureWarni
ng: The default value of numeric_only in DataFrame.corr is deprecated. In a f
uture version, it will default to False. Select only valid columns or specify
the value of numeric_only to silence this warning.
  correlation = df.corr().round(2)
```

Out[21]: <Axes: >

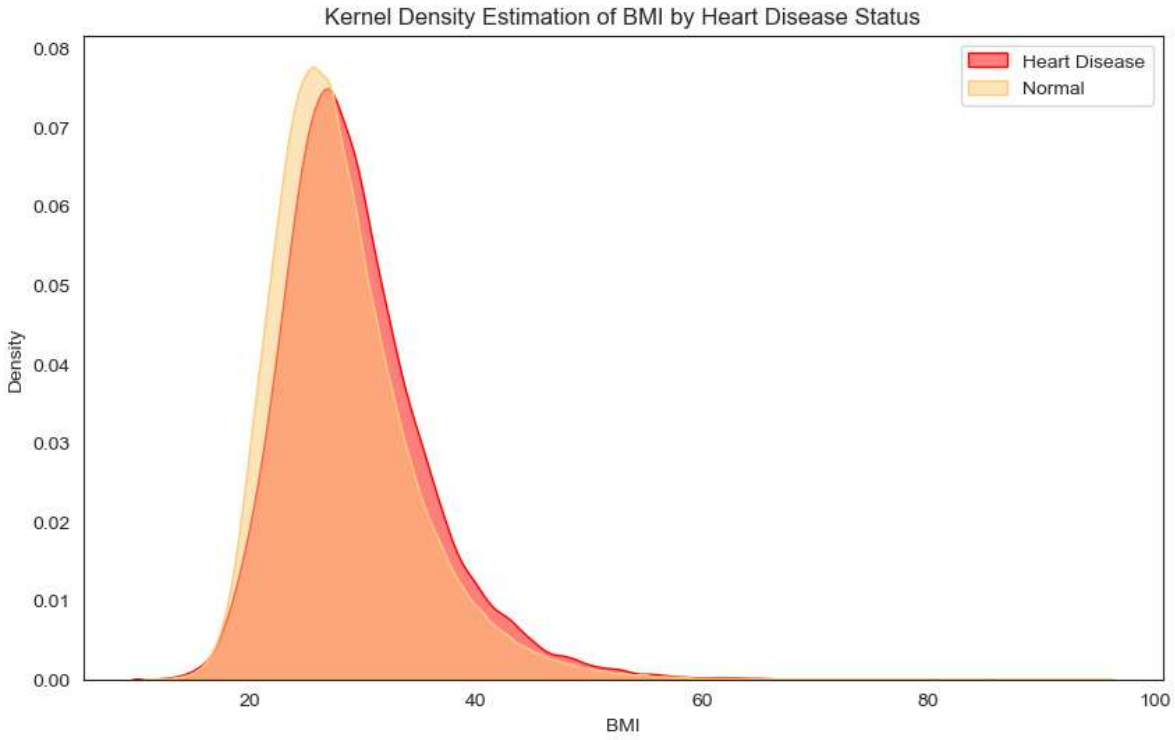


In [22]:
```
sns.set_style('white')
sns.set_palette('YlOrBr')
plt.figure(figsize = (13,6))
plt.title('Distribution of correlation of features')
abs(correlation['HeartDisease']).sort_values()[:-1].plot.barh()
plt.show()
```

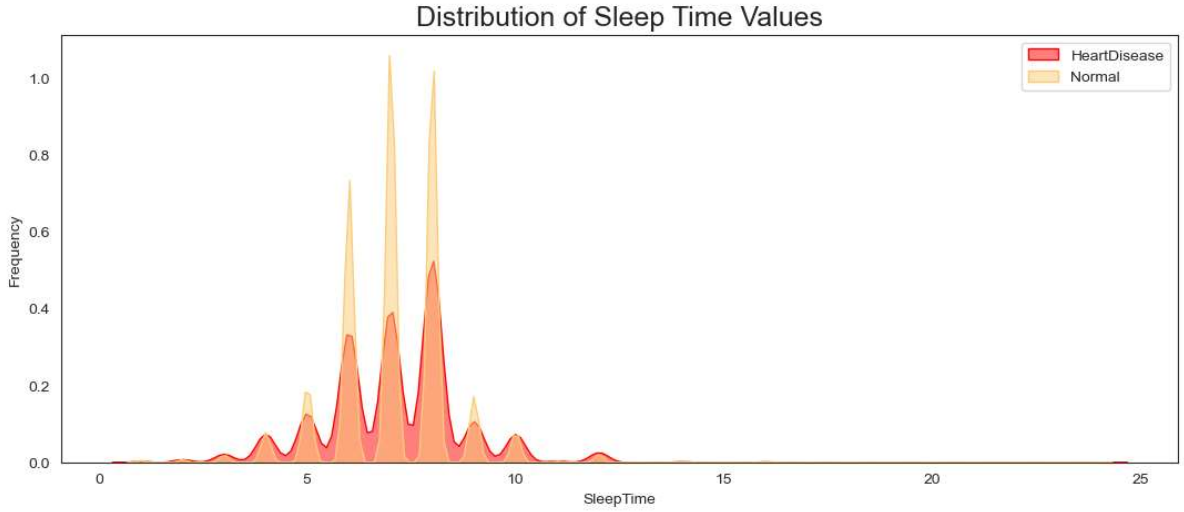


In [91]:

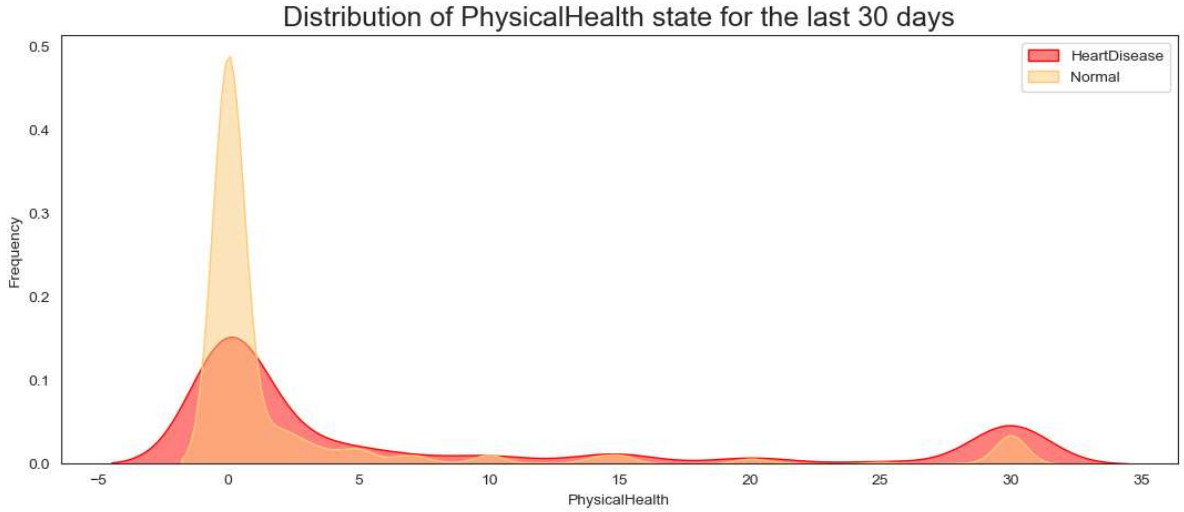
```
fig, ax = plt.subplots(figsize=(10, 6))

sns.kdeplot(df[df["HeartDisease"] == 1]["BMI"], alpha=0.5, fill=True, color="r
sns.kdeplot(df[df["HeartDisease"] == 0]["BMI"], alpha=0.5, fill=True, color="#

ax.set_xlabel("BMI")
ax.set_ylabel("Density")

ax.set_title("Kernel Density Estimation of BMI by Heart Disease Status")
ax.legend()

plt.show()
```



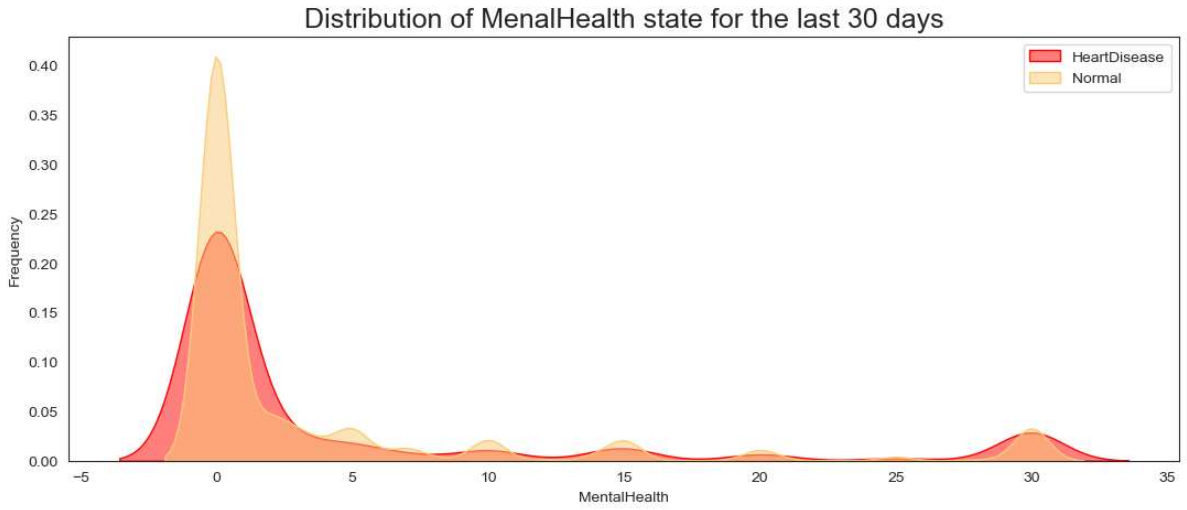
In [93]:
```
fig, ax = plt.subplots(figsize = (13,5))
sns.kdeplot(df[df["HeartDisease"]==1]["SleepTime"], alpha=0.5,fill = True, col
sns.kdeplot(df[df["HeartDisease"]==0]["SleepTime"], alpha=0.5,fill = True, col
ax.set_title('Distribution of Sleep Time Values', fontsize = 18)
ax.set_xlabel("SleepTime")
ax.set_ylabel("Frequency")
ax.legend()
plt.show()
```



In [94]:
```
fig, ax = plt.subplots(figsize = (13,5))
sns.kdeplot(df[df["HeartDisease"]==1]["PhysicalHealth"], alpha=0.5,fill = True
sns.kdeplot(df[df["HeartDisease"]==0]["PhysicalHealth"], alpha=0.5,fill = True
plt.title('Distribution of PhysicalHealth state for the last 30 days', fontsiz
ax.set_xlabel("PhysicalHealth")
ax.set_ylabel("Frequency")
ax.legend()
plt.show()
```



In [95]:
```
fig, ax = plt.subplots(figsize = (13,5))
sns.kdeplot(df[df["HeartDisease"]==1]["MentalHealth"], alpha=0.5,fill = True,
sns.kdeplot(df[df["HeartDisease"]==0]["MentalHealth"], alpha=0.5,fill = True,
plt.title('Distribution of MenalHealth state for the last 30 days', fontsize =
ax.set_xlabel("MentalHealth")
ax.set_ylabel("Frequency")
ax.legend()
plt.show()
```



Split Dataset for Training and Testing

In [8]:
```
#Select Features
features = df.drop(columns =['HeartDisease'], axis = 1)

#Select Target
target = df['HeartDisease']

# Set Training and Testing Data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, target, shuffle


print('Shape of training feature:', X_train.shape)
print('Shape of testing feature:', X_test.shape)
print('Shape of training label:', y_train.shape)
print('Shape of training label:', y_test.shape)
```

```
Shape of training feature: (255836, 17)
Shape of testing feature: (63959, 17)
Shape of training label: (255836,)
Shape of training label: (63959,)
```

Data Preprocessing

In [9]:
```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Create a ColumnTransformer
transformer = make_column_transformer(
    (OneHotEncoder(sparse=False), ['AgeCategory', 'Race', 'GenHealth']),
    remainder='passthrough')

# Encode training data
transformed_train = transformer.fit_transform(X_train)
transformed_train_data = pd.DataFrame(transformed_train, columns=transformer.g

# Concat the two tables for training data
transformed_train_data.reset_index(drop=True, inplace=True)
X_train.reset_index(drop=True, inplace=True)
X_train = pd.concat([transformed_train_data, X_train], axis=1)

# Remove old columns from training data
X_train.drop(['AgeCategory', 'Race', 'GenHealth'], axis=1, inplace=True)

# Encode test data (do not fit again)
transformed_test = transformer.transform(X_test)
transformed_test_data = pd.DataFrame(transformed_test, columns=transformer.get

# Concat the two tables for test data
transformed_test_data.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
X_test = pd.concat([transformed_test_data, X_test], axis=1)

# Remove old columns from test data
X_test.drop(['AgeCategory', 'Race', 'GenHealth'], axis=1, inplace=True)
```

```
C:\Users\Asus\anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:
828: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 an
d will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse
` to its default value.
  warnings.warn(
```

In [10]:
```
scaler = StandardScaler()

# Scale trainint data
X_train = scaler.fit_transform(X_train)

# Scale test data
X_test = scaler.fit_transform(X_test)
```

In [34]:
```
def evaluate_model(model, x_test, y_test):
    from sklearn import metrics

    # Predict Test Data
    y_pred = model.predict(x_test)

    # Calculate accuracy, precision, recall, f1-score, and kappa score
    acc = metrics.accuracy_score(y_test, y_pred)
    prec = metrics.precision_score(y_test, y_pred)
    rec = metrics.recall_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred)
    kappa = metrics.cohen_kappa_score(y_test, y_pred)

    # Calculate area under curve (AUC)
    y_pred_proba = model.predict_proba(x_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
    auc = metrics.roc_auc_score(y_test, y_pred_proba)

    # Display confussion matrix
    cm = metrics.confusion_matrix(y_test, y_pred)

    return {'acc': acc, 'prec': prec, 'rec': rec, 'f1': f1, 'kappa': kappa,
            'fpr': fpr, 'tpr': tpr, 'auc': auc, 'cm': cm}
```

In [35]:
```
X_train
```

Out[35]:
```
array([[-0.26549634, -0.23664263, -0.25032481, ..., -0.39281938,
        -0.19512091, -0.32147374],
       [-0.26549634, -0.23664263, -0.25032481, ..., -0.39281938,
        -0.19512091, -0.32147374],
       [-0.26549634, -0.23664263, -0.25032481, ..., -0.39281938,
        -0.19512091, -0.32147374],
       ...,
       [-0.26549634, -0.23664263, -0.25032481, ..., -0.39281938,
        -0.19512091, -0.32147374],
       [-0.26549634, -0.23664263, -0.25032481, ...,  2.54569927,
        -0.19512091, -0.32147374],
       [-0.26549634,  4.22578124, -0.25032481, ..., -0.39281938,
        -0.19512091, -0.32147374]])
```

In [36]:
```
# Build the Logistic Regression model
logreg = LogisticRegression()

# Train the model
logreg.fit(X_train, y_train)

# Evaluate the Logistic Regression model
logreg_eval = evaluate_model(logreg, X_test, y_test)

# Print evaluation results
print('Logistic Regression Model Evaluation:')
print('Accuracy:', logreg_eval['acc'])
print('Precision:', logreg_eval['prec'])
print('Recall:', logreg_eval['rec'])
print('F1 Score:', logreg_eval['f1'])
print('Cohen\'s Kappa Score:', logreg_eval['kappa'])
print('Area Under Curve:', logreg_eval['auc'])
print('Confusion Matrix:\n', logreg_eval['cm'])
```

```
Logistic Regression Model Evaluation:
Accuracy: 0.9161181381822731
Precision: 0.5367847411444142
Recall: 0.1085200146896805
F1 Score: 0.1805407056667176
Cohen's Kappa Score: 0.1563806759014693
Area Under Curve: 0.8401338964390077
Confusion Matrix:
 [[58003   510]
 [ 4855   591]]
```

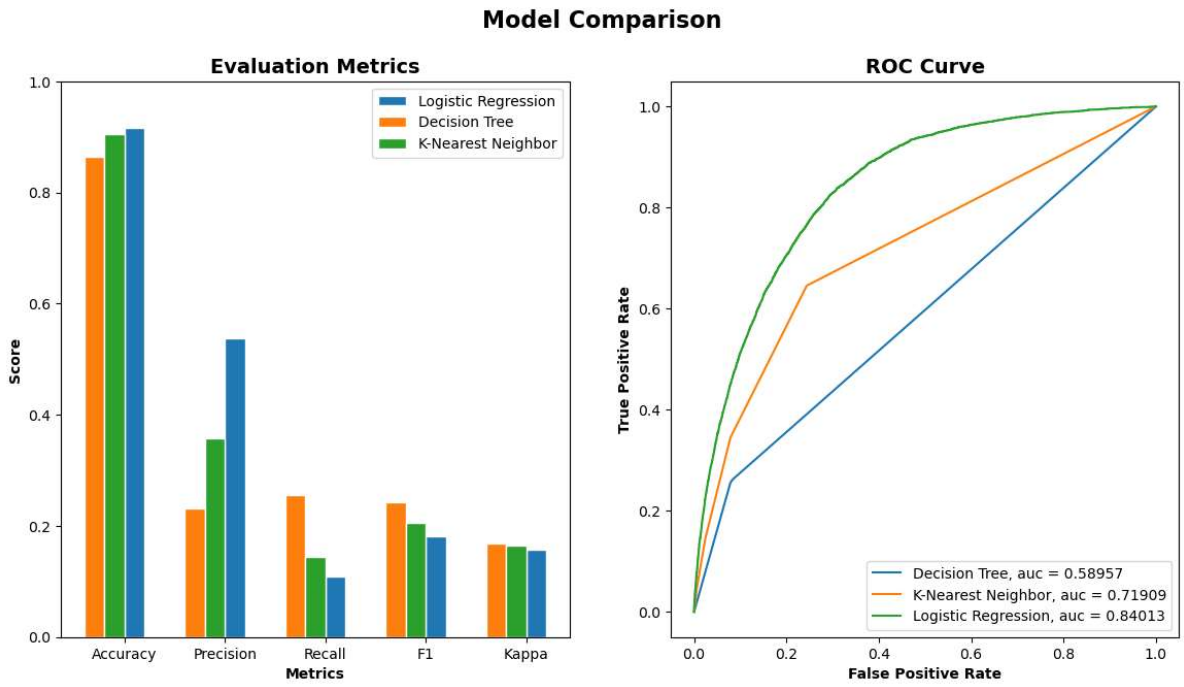
In [37]:
```
# Building a model using KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)

knn.fit(X_train, y_train)


# Evaluate Model
knn_eval = evaluate_model(knn, X_test, y_test)

# Print result
print('Accuracy:', knn_eval['acc'])
print('Precision:', knn_eval['prec'])
print('Recall:', knn_eval['rec'])
print('F1 Score:', knn_eval['f1'])
print('Cohens Kappa Score:', knn_eval['kappa'])
print('Area Under Curve:', knn_eval['auc'])
print('Confusion Matrix:\n', knn_eval['cm'])
```

```
Accuracy: 0.9050016416766992
Precision: 0.3570780399274047
Recall: 0.14450973191333089
F1 Score: 0.2057516339869281
Cohens Kappa Score: 0.16477268538693723
Area Under Curve: 0.7190876830488478
Confusion Matrix:
 [[57096  1417]
 [ 4659   787]]
```

In [38]:
```
# Building Decision Tree model
clf = tree.DecisionTreeClassifier(random_state=0)
clf.fit(X_train, y_train)

# Evaluate Model
clf_eval = evaluate_model(clf, X_test, y_test)

# Print result
print('Accuracy:', clf_eval['acc'])
print('Precision:', clf_eval['prec'])
print('Recall:', clf_eval['rec'])
print('F1 Score:', clf_eval['f1'])
print('Cohens Kappa Score:', clf_eval['kappa'])
print('Area Under Curve:', clf_eval['auc'])
print('Confusion Matrix:\n', clf_eval['cm'])# Evaluate Model
```

```
Accuracy: 0.8646945699588799
Precision: 0.232220367278798
Recall: 0.25541681968417185
F1 Score: 0.2432668765302553
Cohens Kappa Score: 0.1691567154170338
Area Under Curve: 0.589573855978808
Confusion Matrix:
 [[53914  4599]
 [ 4055  1391]]
```

In [39]:
```
# Intitialize figure with two plots
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Model Comparison', fontsize=16, fontweight='bold')
fig.set_figheight(7)
fig.set_figwidth(14)
fig.set_facecolor('white')

# First plot
## set bar size
barWidth = 0.2
clf_score = [clf_eval['acc'], clf_eval['prec'], clf_eval['rec'], clf_eval['f1'
knn_score = [knn_eval['acc'], knn_eval['prec'], knn_eval['rec'], knn_eval['f1'
logreg_score = [logreg_eval['acc'],logreg_eval['prec'],logreg_eval['rec'],logr

## Set position of bar on X axis
r1 = np.arange(len(clf_score))
r2 = [x + barWidth for x in r1]
r3 = [x+ barWidth for x in r2]
## Make the plot
ax1.bar(r3,logreg_score,width=barWidth,edgecolor='white',label='Logistic Regre
ax1.bar(r1, clf_score, width=barWidth, edgecolor='white', label='Decision Tree
ax1.bar(r2, knn_score, width=barWidth, edgecolor='white', label='K-Nearest Nei

## Configure x and y axis
ax1.set_xlabel('Metrics', fontweight='bold')
labels = ['Accuracy', 'Precision', 'Recall', 'F1', 'Kappa']
ax1.set_xticks([r + (barWidth * 1.5) for r in range(len(clf_score))], )
ax1.set_xticklabels(labels)
ax1.set_ylabel('Score', fontweight='bold')
ax1.set_ylim(0, 1)

## Create legend & title
ax1.set_title('Evaluation Metrics', fontsize=14, fontweight='bold')
ax1.legend()



# Second plot
## Comparing ROC Curve
ax2.plot(clf_eval['fpr'], clf_eval['tpr'], label='Decision Tree, auc = {:0.5f}
ax2.plot(knn_eval['fpr'], knn_eval['tpr'], label='K-Nearest Neighbor, auc = {:
ax2.plot(logreg_eval['fpr'], logreg_eval['tpr'], label='Logistic Regression, a

## Configure x and y axis
ax2.set_xlabel('False Positive Rate', fontweight='bold')
ax2.set_ylabel('True Positive Rate', fontweight='bold')

## Create legend & title
ax2.set_title('ROC Curve', fontsize=14, fontweight='bold')
ax2.legend(loc=4)

plt.show()
```



If we want to prioritize precision, Logistic Regression might be preferred due to its higher precision value. If recall is important, Decision Tree has the highest recall value. The AUC values indicate that Logistic Regression has the highest discrimination ability.

Ultimately, the choice depends on our specific use case and what trade-offs we're willing to make between precision, recall, and other metrics.