Random forest model on stroke prediction

My model predicts whether a person has had a stroke or not based on the input features.

It classifies individuals into two categories:

1. Stroke (1): The model predicts that the person has had a stroke.
2. No Stroke (0): The model predicts that the person has not had a stroke.

The prediction is based on the features provided to the model during inference. These features might include information such as gender, age, hypertension, heart disease, marital status, work type, residence type, average glucose level, BMI, and smoking status. The model analyzes these features and makes a prediction based on the patterns it has learned from the training data.

Loading data

In [3]:
```
import pandas as pd

file_path = r'C:\Users\roari\Downloads\Brain.csv'
data = pd.read_csv(file_path)

data.head()
```

Out[3]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glu
0	Male	67.0	0	1	Yes	Private	Urban	
1	Male	80.0	0	1	Yes	Private	Rural	
2	Female	49.0	0	0	Yes	Private	Urban	
3	Female	79.0	1	0	Yes	Self-employed	Rural	
4	Male	81.0	0	0	Yes	Private	Urban	

Head

In [4]:
```
data.head(5)
```

Out[4]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glu
0	Male	67.0	0	1	Yes	Private	Urban	
1	Male	80.0	0	1	Yes	Private	Rural	
2	Female	49.0	0	0	Yes	Private	Urban	
3	Female	79.0	1	0	Yes	Self-employed	Rural	
4	Male	81.0	0	0	Yes	Private	Urban	

Tail

In [5]:
```
data.tail(5)
```

Out[5]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_
4976	Male	41.0	0	0	No	Private	Rural	
4977	Male	40.0	0	0	Yes	Private	Urban	
4978	Female	45.0	1	0	Yes	Govt_job	Rural	
4979	Male	40.0	0	0	Yes	Private	Rural	
4980	Female	80.0	1	0	Yes	Private	Urban	

Description

In [7]:
```
descriptive_stats = data.describe(include='all')
descriptive_stats
```

Out[7]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type
count	4981	4981.000000	4981.000000	4981.000000	4981	4981	4981
nique	2	NaN	NaN	NaN	2	4	2
top	Female	NaN	NaN	NaN	Yes	Private	Urban
freq	2907	NaN	NaN	NaN	3280	2860	2532
mean	NaN	43.419859	0.096165	0.055210	NaN	NaN	NaN
std	NaN	22.662755	0.294848	0.228412	NaN	NaN	NaN
min	NaN	0.080000	0.000000	0.000000	NaN	NaN	NaN
25%	NaN	25.000000	0.000000	0.000000	NaN	NaN	NaN
50%	NaN	45.000000	0.000000	0.000000	NaN	NaN	NaN
75%	NaN	61.000000	0.000000	0.000000	NaN	NaN	NaN
max	NaN	82.000000	1.000000	1.000000	NaN	NaN	NaN

Information

In [10]:
```
data_info = data.info()
print(data_info)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4981 entries, 0 to 4980
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             4981 non-null   object 
 1   age                4981 non-null   float64
 2   hypertension       4981 non-null   int64  
 3   heart_disease      4981 non-null   int64  
 4   ever_married       4981 non-null   object 
 5   work_type          4981 non-null   object 
 6   Residence_type     4981 non-null   object 
 7   avg_glucose_level  4981 non-null   float64
 8   bmi                4981 non-null   float64
 9   smoking_status     4981 non-null   object 
 10  stroke             4981 non-null   int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 428.2+ KB
None
```

Null

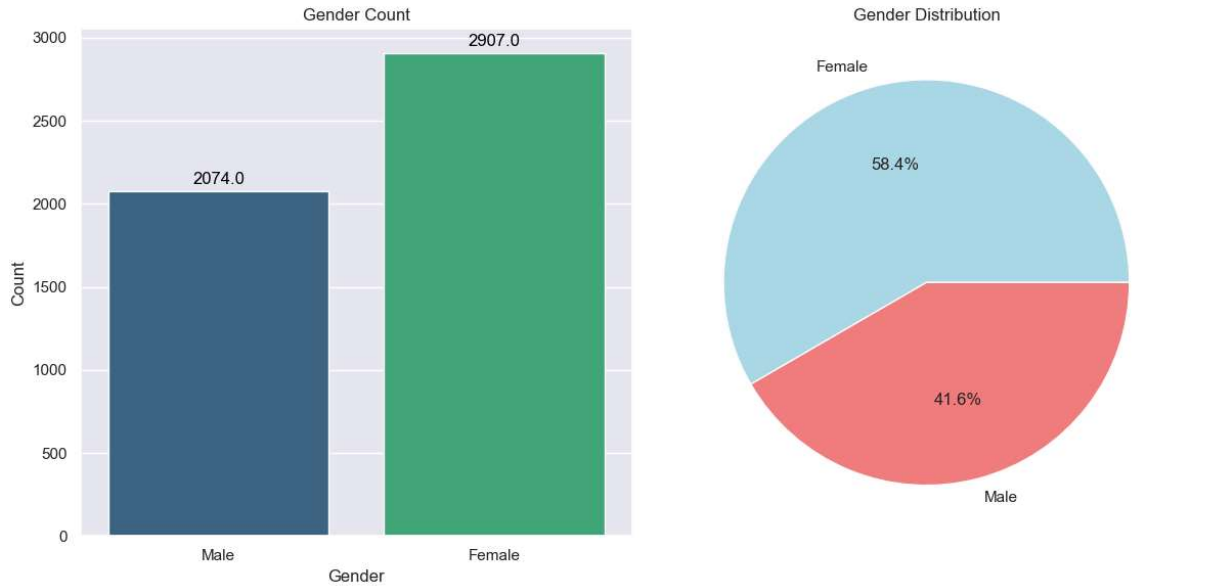
In [17]:
```
missing_values = data.isnull().sum()

print("Null Values in the Dataset:")
print(missing_values)
```

```
Null Values in the Dataset:
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

Exploring dataset

Understanding Gender

In [14]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
ax1 = sns.countplot(x='gender', data=data, palette='viridis')

for p in ax1.patches:
    ax1.annotate(f'{p.get_height()}',
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='baseline',
                 fontsize=12, color='black', xytext=(0, 5),
                 textcoords='offset points')

plt.title('Gender Count')
plt.xlabel('Gender')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
gender_counts = data['gender'].value_counts()
labels = gender_counts.index
plt.pie(gender_counts, labels=labels, autopct='%1.1f%%', colors=['lightblue',
plt.title('Gender Distribution')

plt.tight_layout()

plt.show()
```



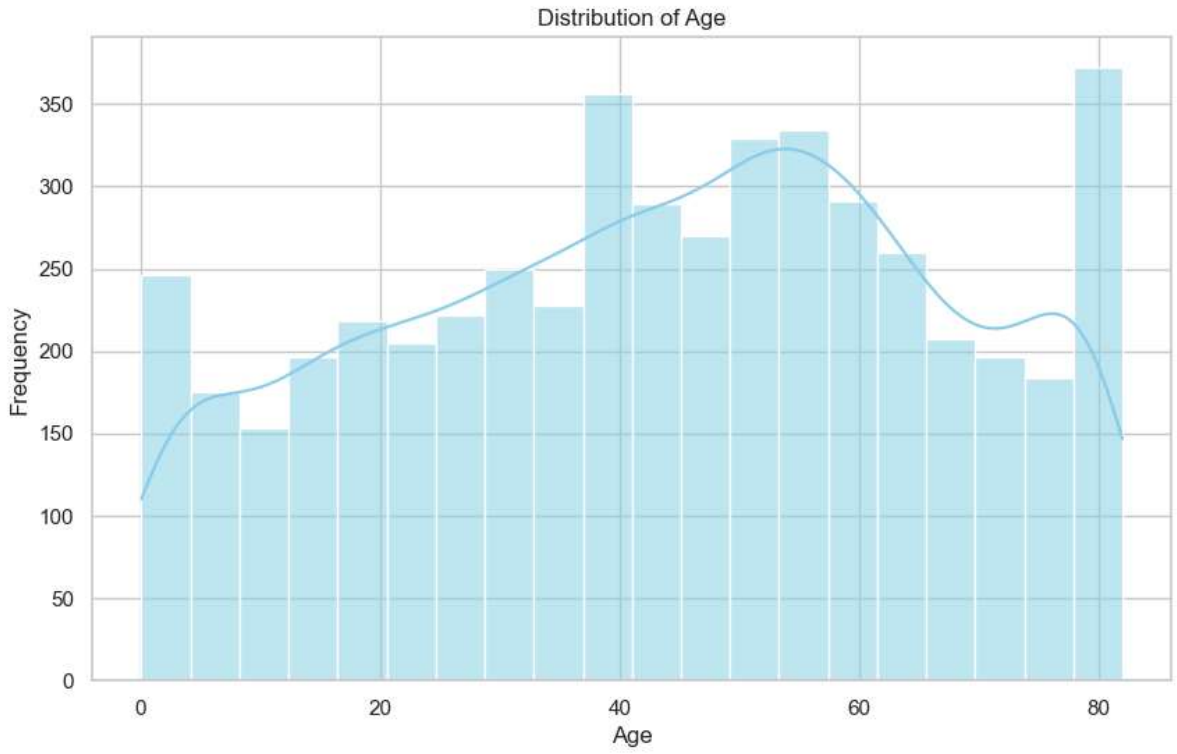
In [16]:
```
import pandas as pd

gender_counts = data['gender'].value_counts().reset_index()
gender_counts.columns = ['Gender', 'Count']

print("Gender Counts:")
print(gender_counts)
```

```
Gender Counts:
   Gender  Count
0  Female   2907
1    Male   2074
```

Understanding Age

In [18]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="whitegrid")

plt.figure(figsize=(10, 6))
sns.histplot(data['age'], kde=True, color='skyblue')

plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')

plt.show()
```

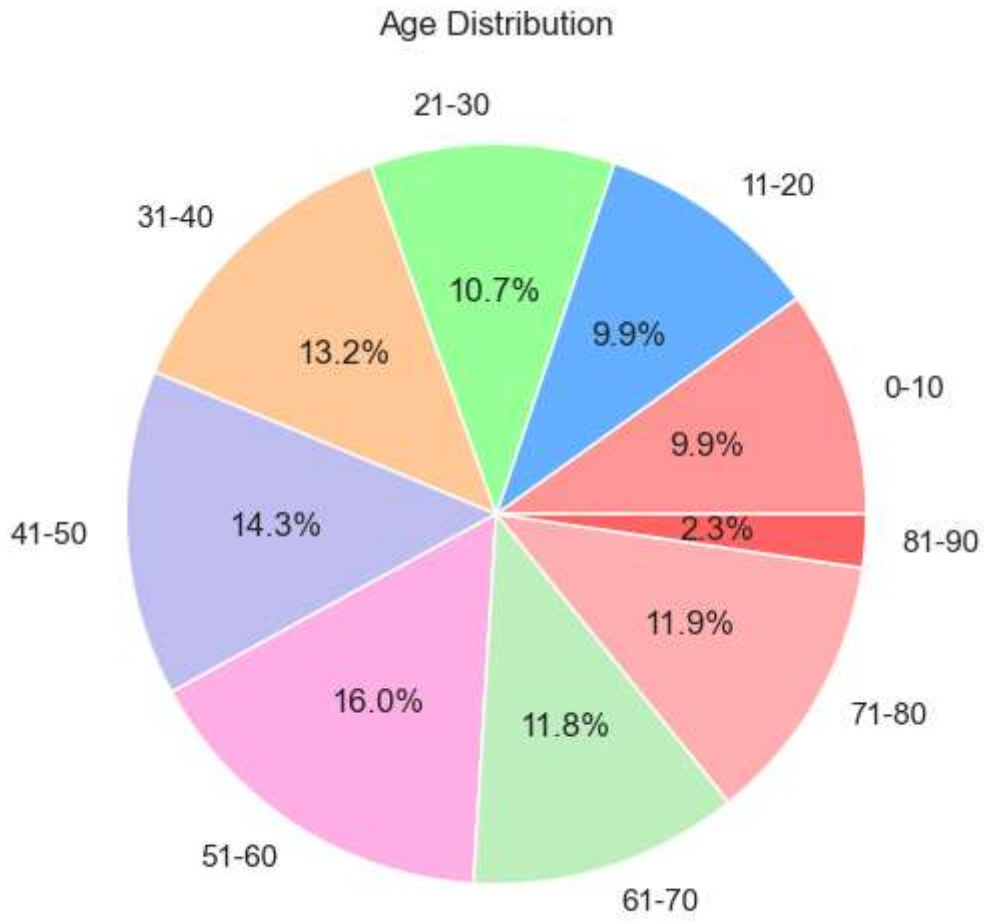
```
C:\Users\roari\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWa
rning: use_inf_as_na option is deprecated and will be removed in a future ver
sion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [154]:
```
def map_age_group(age):
    if age <= 10:
        return '0-10'
    elif 11 <= age <= 20:
        return '11-20'
    elif 21 <= age <= 30:
        return '21-30'
    elif 31 <= age <= 40:
        return '31-40'
    elif 41 <= age <= 50:
        return '41-50'
    elif 51 <= age <= 60:
        return '51-60'
    elif 61 <= age <= 70:
        return '61-70'
    elif 71 <= age <= 80:
        return '71-80'
    elif 81 <= age <= 90:
        return '81-90'
    else:
        return '90+'

age_counts_sorted['Age Group'] = age_counts_sorted['Age'].apply(map_age_group)

age_counts_grouped = age_counts_sorted.groupby('Age Group', as_index=False)['C

print("Grouped Age Counts:")
print(age_counts_grouped)
```

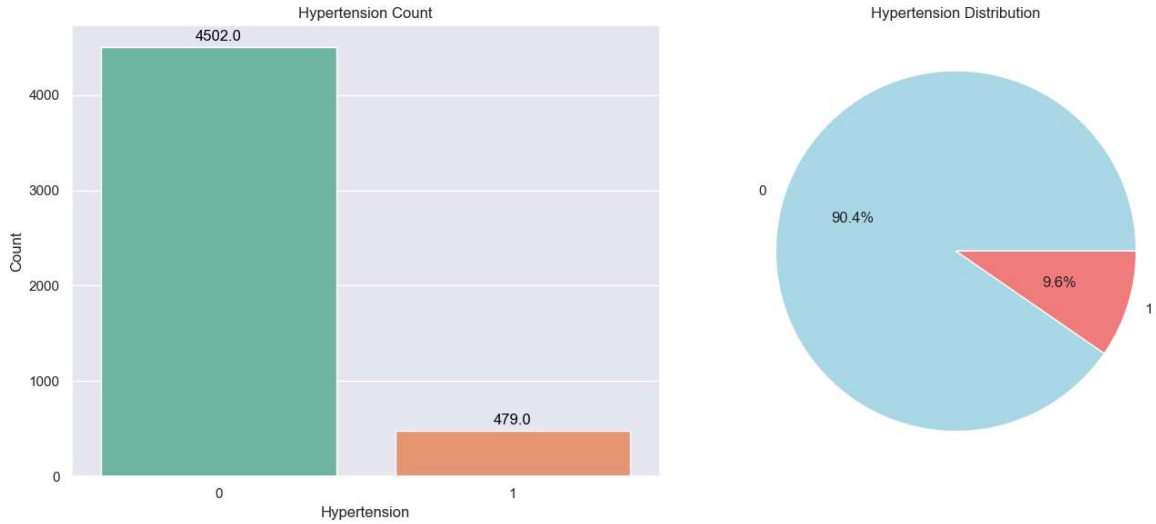
```
Grouped Age Counts:
  Age Group  Count
0      0-10    493
1     11-20    495
2     21-30    532
3     31-40    658
4     41-50    710
5     51-60    799
6     61-70    587
7     71-80    591
8     81-90    116
```

In [158]:
```
import matplotlib.pyplot as plt

colors = ['#ff9999', '#66b3ff', '#99ff99', '#ffcc99', '#c2c2f0', '#ffb3e6', '#

plt.figure(figsize=(8, 6))
plt.pie(age_counts_grouped['Count'], labels=age_counts_grouped['Age Group'], a
plt.title('Age Distribution')
plt.show()
```



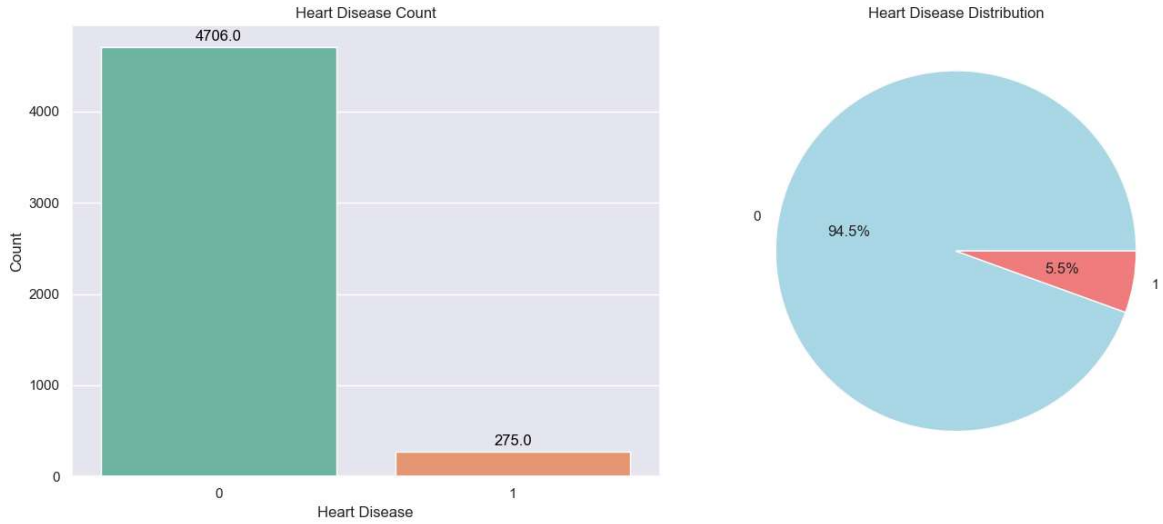
Understanding Hypertension

In [23]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
ax1 = sns.countplot(x='hypertension', data=data, palette='Set2')

for p in ax1.patches:
    ax1.annotate(f'{p.get_height()}',
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='baseline',
                 fontsize=12, color='black', xytext=(0, 5),
                 textcoords='offset points')

plt.title('Hypertension Count')
plt.xlabel('Hypertension')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
hypertension_counts = data['hypertension'].value_counts()
labels = hypertension_counts.index
plt.pie(hypertension_counts, labels=labels, autopct='%1.1f%%', colors=['lightb
plt.title('Hypertension Distribution')

plt.tight_layout()

plt.show()
```



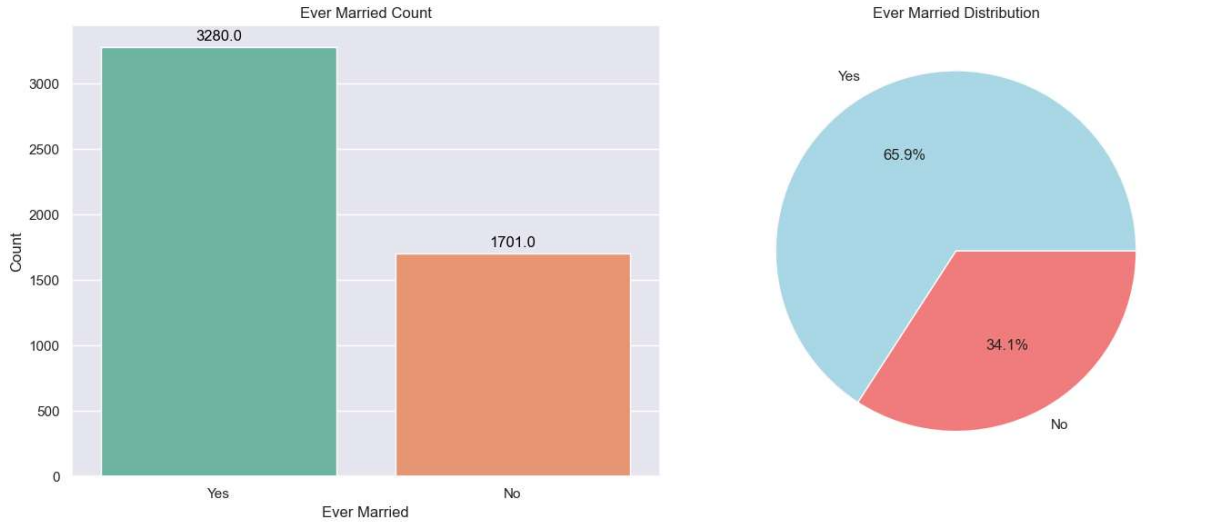
Understanding Heart Disease

In [24]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
ax1 = sns.countplot(x='heart_disease', data=data, palette='Set2')

for p in ax1.patches:
    ax1.annotate(f'{p.get_height()}',
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='baseline',
                 fontsize=12, color='black', xytext=(0, 5),
                 textcoords='offset points')

plt.title('Heart Disease Count')
plt.xlabel('Heart Disease')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
heart_disease_counts = data['heart_disease'].value_counts()
labels = heart_disease_counts.index
plt.pie(heart_disease_counts, labels=labels, autopct='%1.1f%%', colors=['light
plt.title('Heart Disease Distribution')

plt.tight_layout()

plt.show()
```



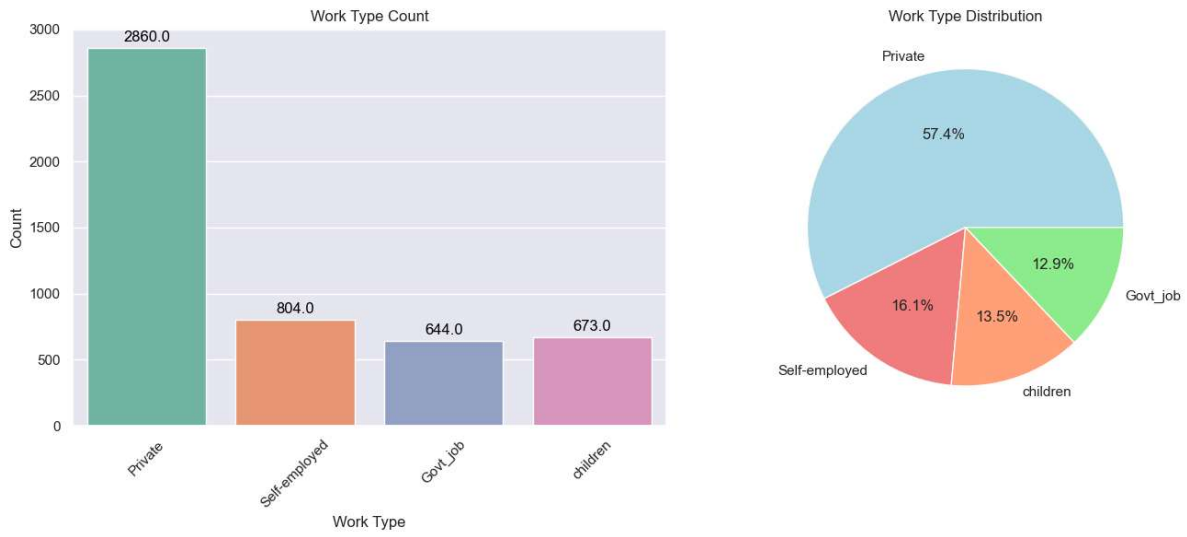
Married Status

In [25]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
ax1 = sns.countplot(x='ever_married', data=data, palette='Set2')

for p in ax1.patches:
    ax1.annotate(f'{p.get_height()}',
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='baseline',
                 fontsize=12, color='black', xytext=(0, 5),
                 textcoords='offset points')

plt.title('Ever Married Count')
plt.xlabel('Ever Married')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
ever_married_counts = data['ever_married'].value_counts()
labels = ever_married_counts.index
plt.pie(ever_married_counts, labels=labels, autopct='%1.1f%%', colors=['lightb
plt.title('Ever Married Distribution')

plt.tight_layout()

plt.show()
```



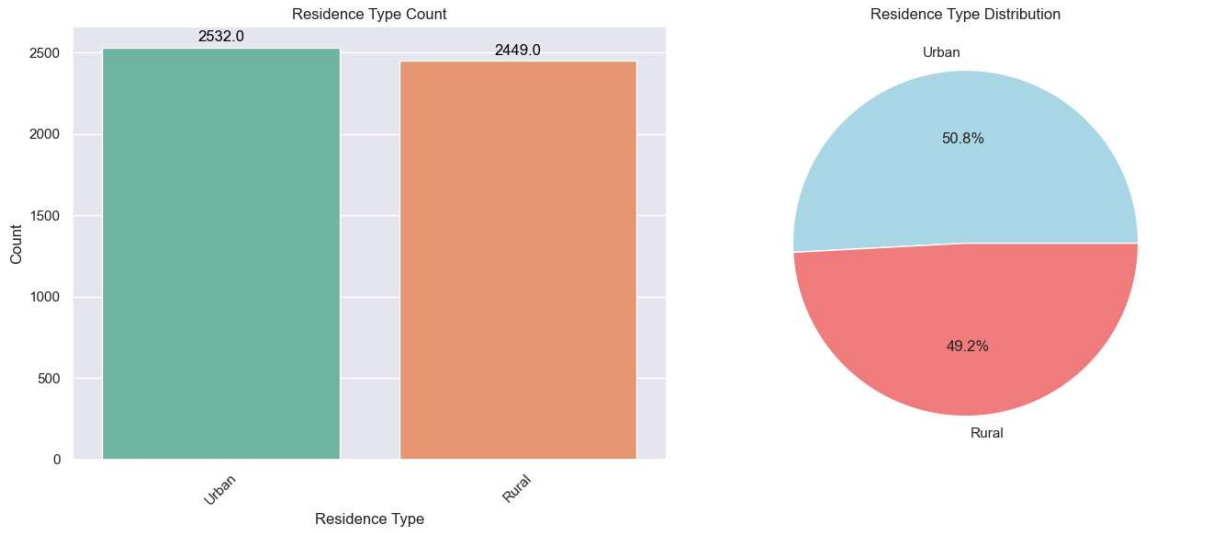
Understanding Work Type

In [26]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
ax1 = sns.countplot(x='work_type', data=data, palette='Set2')

for p in ax1.patches:
    ax1.annotate(f'{p.get_height()}',
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='baseline',
                 fontsize=12, color='black', xytext=(0, 5),
                 textcoords='offset points')

plt.title('Work Type Count')
plt.xlabel('Work Type')
plt.ylabel('Count')

plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
work_type_counts = data['work_type'].value_counts()
labels = work_type_counts.index
plt.pie(work_type_counts, labels=labels, autopct='%1.1f%%', colors=['lightblue
plt.title('Work Type Distribution')

plt.tight_layout()

plt.show()
```



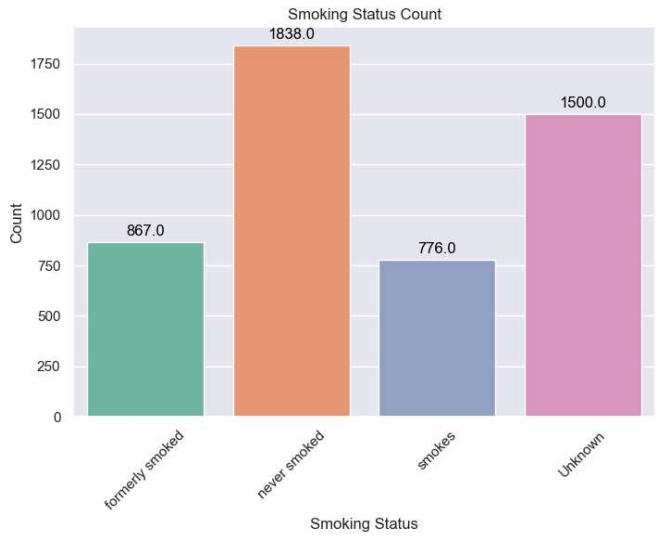
Understanding Residence type

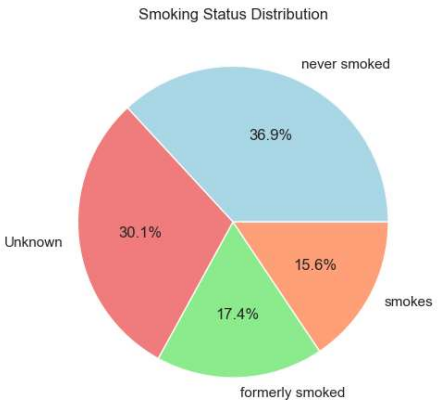
In [27]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
ax1 = sns.countplot(x='Residence_type', data=data, palette='Set2')

for p in ax1.patches:
    ax1.annotate(f'{p.get_height()}',
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='baseline',
                 fontsize=12, color='black', xytext=(0, 5),
                 textcoords='offset points')

plt.title('Residence Type Count')
plt.xlabel('Residence Type')
plt.ylabel('Count')

plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
residence_type_counts = data['Residence_type'].value_counts()
labels = residence_type_counts.index
plt.pie(residence_type_counts, labels=labels, autopct='%1.1f%%', colors=['ligh
plt.title('Residence Type Distribution')

plt.tight_layout()

plt.show()
```



Understanding Smoking Status

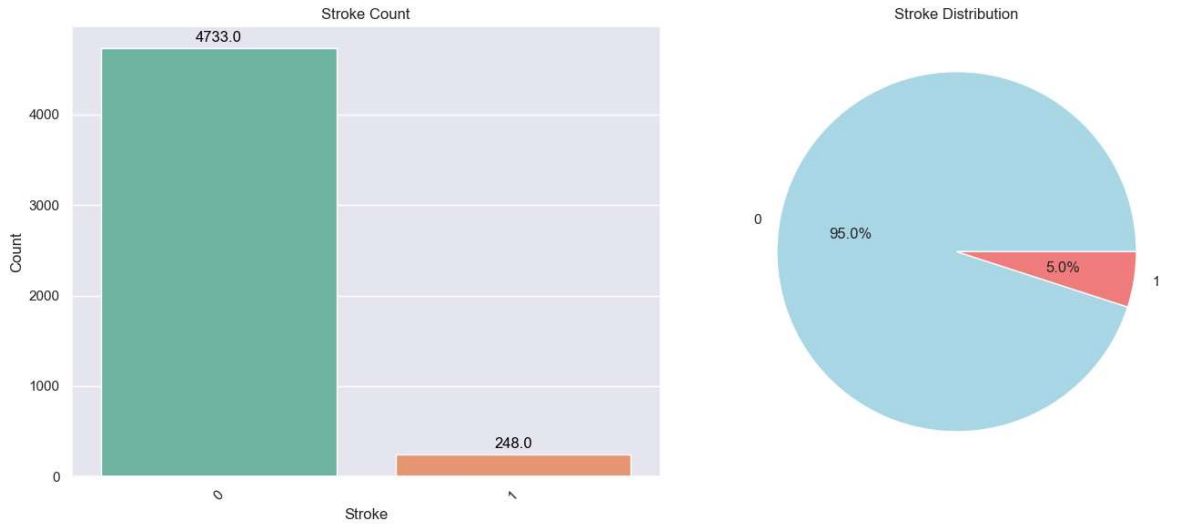
In [28]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
ax1 = sns.countplot(x='smoking_status', data=data, palette='Set2')

for p in ax1.patches:
    ax1.annotate(f'{p.get_height()}',
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='baseline',
                 fontsize=12, color='black', xytext=(0, 5),
                 textcoords='offset points')

plt.title('Smoking Status Count')
plt.xlabel('Smoking Status')
plt.ylabel('Count')

plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
smoking_status_counts = data['smoking_status'].value_counts()
labels = smoking_status_counts.index
plt.pie(smoking_status_counts, labels=labels, autopct='%1.1f%%', colors=['ligh
plt.title('Smoking Status Distribution')

plt.tight_layout()

plt.show()
```





Understanding Smoker count

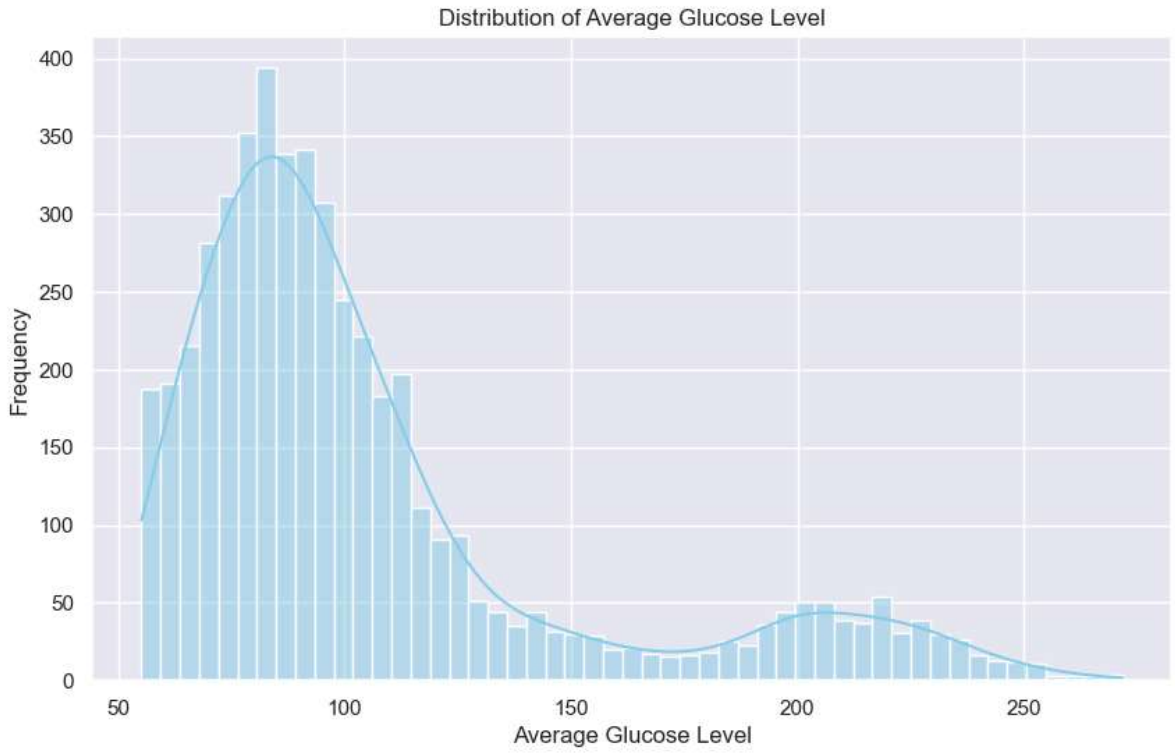
In [29]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
ax1 = sns.countplot(x='stroke', data=data, palette='Set2')

for p in ax1.patches:
    ax1.annotate(f'{p.get_height()}',
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='baseline',
                 fontsize=12, color='black', xytext=(0, 5),
                 textcoords='offset points')

plt.title('Stroke Count')
plt.xlabel('Stroke')
plt.ylabel('Count')

plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
stroke_counts = data['stroke'].value_counts()
labels = stroke_counts.index
plt.pie(stroke_counts, labels=labels, autopct='%1.1f%%', colors=['lightblue',
plt.title('Stroke Distribution')

plt.tight_layout()

plt.show()
```



Understanding Distribution of Glucose level

In [30]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.figure(figsize=(10, 6))
sns.histplot(data['avg_glucose_level'], kde=True, color='skyblue')

plt.title('Distribution of Average Glucose Level')
plt.xlabel('Average Glucose Level')
plt.ylabel('Frequency')

plt.show()
```

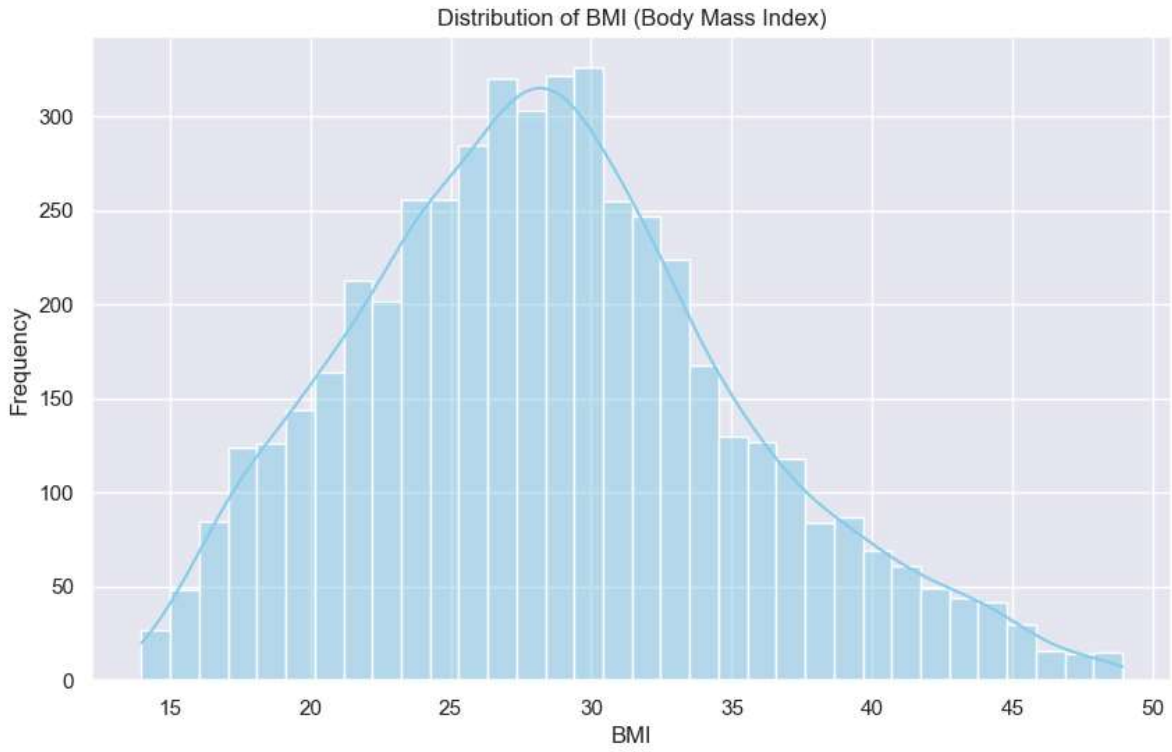
```
C:\Users\roari\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWa
rning: use_inf_as_na option is deprecated and will be removed in a future ver
sion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Understanding BMI

In [31]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="darkgrid")

plt.figure(figsize=(10, 6))
sns.histplot(data['bmi'], kde=True, color='skyblue')

plt.title('Distribution of BMI (Body Mass Index)')
plt.xlabel('BMI')
plt.ylabel('Frequency')

plt.show()
```

```
C:\Users\roari\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWa
rning: use_inf_as_na option is deprecated and will be removed in a future ver
sion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Random Forest model

Import Libraries and Load Data

In [14]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_
import seaborn as sns
import matplotlib.pyplot as plt

file_path = "C:/Users/roari/Downloads/Brain.csv"
data = pd.read_csv(file_path)

data.head()
```

Out[14]:

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glu
0	Male	67.0	0	1	Yes	Private	Urban	
1	Male	80.0	0	1	Yes	Private	Rural	
2	Female	49.0	0	0	Yes	Private	Urban	
3	Female	79.0	1	0	Yes	Self-employed	Rural	
4	Male	81.0	0	0	Yes	Private	Urban	

Data Preprocessing, Oversampling minority class and Spliting training and testing dataset

In [15]:
```
target = 'stroke'
X = data.drop(columns=[target])
y = data[target]

X = pd.get_dummies(X)

X.fillna(X.mean(), inplace=True)

smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X, y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

Create and Train the Random Forest Model

In [16]:
```
rf_model = RandomForestClassifier(n_estimators=200, criterion='gini', random_s

rf_model.fit(X_train, y_train)
```

Out[16]:
```
RandomForestClassifier

RandomForestClassifier(n_estimators=200, random_state=42)
```

Evaluate the Model

In [42]:
```
train_accuracy = accuracy_score(y_train, rf_model.predict(X_train))
test_accuracy = accuracy_score(y_test, rf_model.predict(X_test))

print("Training Accuracy (Gini):", train_accuracy)
print("Testing Accuracy (Gini):", test_accuracy)

train_report = classification_report(y_train, rf_model.predict(X_train))
test_report = classification_report(y_test, rf_model.predict(X_test))

print("*"*125)

print("Training Classification Report (Gini):\n", train_report)

print("*"*125)

print("Testing Classification Report (Gini):\n", test_report)

train_cm = confusion_matrix(y_train, rf_model.predict(X_train))
test_cm = confusion_matrix(y_test, rf_model.predict(X_test))

print("*"*125)

print("Training Confusion Matrix (Gini):\n", train_cm)

print("*"*125)

print("Testing Confusion Matrix (Gini):\n", test_cm)
```

```
Training Accuracy (Gini): 1.0
Testing Accuracy (Gini): 0.9688489968321014
********************************************************************************
**********************************************
Training Classification Report (Gini):
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      3787
           1       1.00      1.00      1.00      3785

    accuracy                           1.00      7572
   macro avg       1.00      1.00      1.00      7572
weighted avg       1.00      1.00      1.00      7572

********************************************************************************
**********************************************
Testing Classification Report (Gini):
               precision    recall  f1-score   support

           0       0.95      0.99      0.97       946
           1       0.99      0.95      0.97       948

    accuracy                           0.97      1894
   macro avg       0.97      0.97      0.97      1894
weighted avg       0.97      0.97      0.97      1894

********************************************************************************
**********************************************
Training Confusion Matrix (Gini):
 [[3787    0]
 [   0 3785]]
********************************************************************************
**********************************************
Testing Confusion Matrix (Gini):
 [[937   9]
 [ 50 898]]
```

Perform Cross-Validation

In [18]:
```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_scorel_score(rf_model, X, y, cv=10, scoring='accuracy')

print("Cross-Validation Scores: ", cv_scores)
print("Mean Cross-Validation Score: ", cv_scores.mean())
```

```
Cross-Validation Scores:  [0.76768743 0.9904963  0.99577614 0.99155227 0.9936
642  0.99894403
 0.99048626 0.99048626 0.99154334 0.98414376]
Mean Cross-Validation Score:  0.9694779999598152
```

Each fold provides a score, typically accuracy in this case, which gives an indication of how well the model is likely to perform on unseen data.

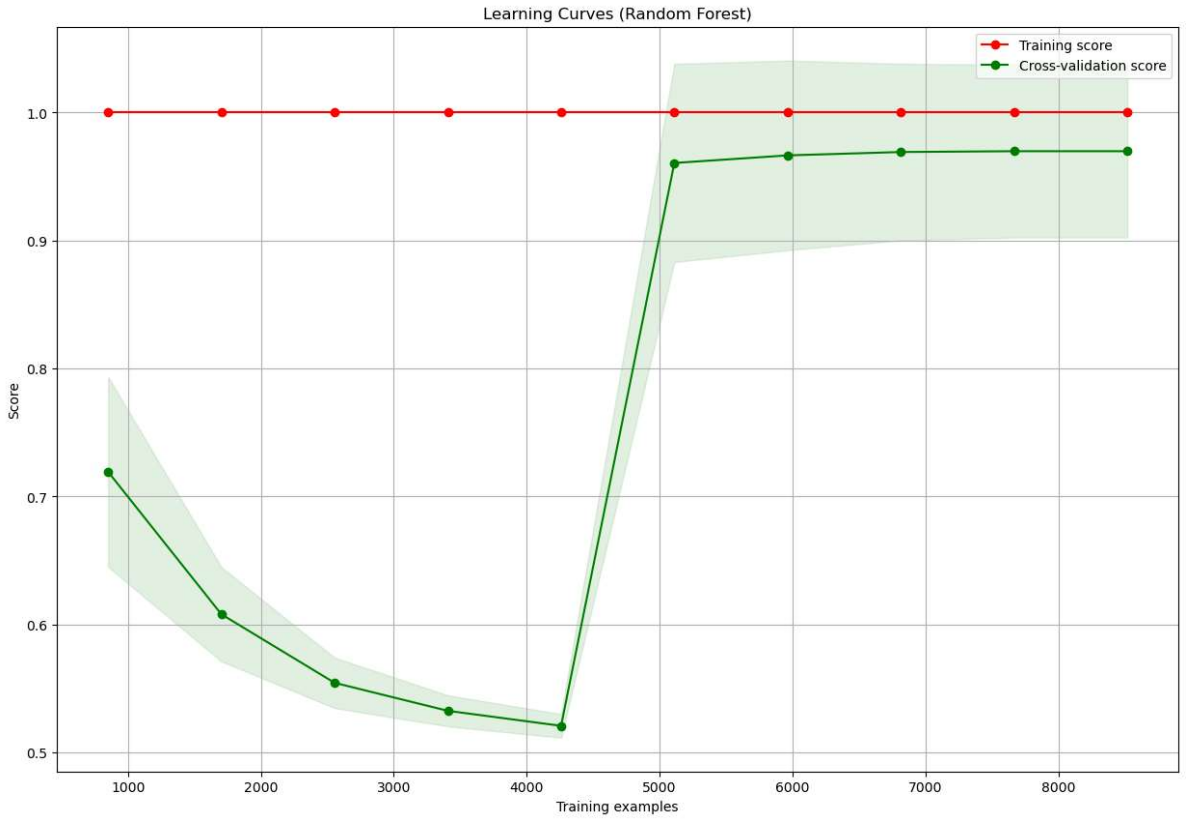
Introduce Regularization

In [43]:
```
rf_model_reg = RandomForestClassifier(n_estimators=200, max_depth=10, random_s

rf_model_reg.fit(X_train, y_train)

train_accuracy_reg = accuracy_score(y_train, rf_model_reg.predict(X_train))
test_accuracy_reg = accuracy_score(y_test, rf_model_reg.predict(X_test))

print("Training Accuracy (Reg):", train_accuracy_reg)
print("Testing Accuracy (Reg):", test_accuracy_reg)

train_report_reg = classification_report(y_train, rf_model_reg.predict(X_train
test_report_reg = classification_report(y_test, rf_model_reg.predict(X_test))

train_cm_reg = confusion_matrix(y_train, rf_model_reg.predict(X_train))
test_cm_reg = confusion_matrix(y_test, rf_model_reg.predict(X_test))

print("*"*125)

print("Training Classification Report (Reg):\n", train_report_reg)

print("*"*125)

print("Testing Classification Report (Reg):\n", test_report_reg)

print("*"*125)

print("Training Confusion Matrix (Reg):\n", train_cm_reg)

print("*"*125)

print("Testing Confusion Matrix (Reg):\n", test_cm_reg)
```

```
Training Accuracy (Reg): 0.9795298468040148
Testing Accuracy (Reg): 0.9640971488912354
**********************************************************************
**********************************************
Training Classification Report (Reg):
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      3787
           1       0.99      0.97      0.98      3785

    accuracy                           0.98      7572
   macro avg       0.98      0.98      0.98      7572
weighted avg       0.98      0.98      0.98      7572

**********************************************************************
**********************************************
Testing Classification Report (Reg):
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       946
           1       0.98      0.95      0.96       948

    accuracy                           0.96      1894
   macro avg       0.96      0.96      0.96      1894
weighted avg       0.96      0.96      0.96      1894

**********************************************************************
**********************************************
Training Confusion Matrix (Reg):
 [[3732   55]
 [ 100 3685]]
**********************************************************************
**********************************************
Testing Confusion Matrix (Reg):
 [[925  21]
 [ 47 901]]
```

Plotting Learning Curves

In [24]:
```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(rf_model, X, y, cv=10,

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.figure(figsize=(15, 10))
plt.title("Learning Curves (Random Forest)")
plt.xlabel("Training examples")
plt.ylabel("Score")
plt.grid()

plt.fill_between(train_sizes, train_scores_mean - train_scores_std, train_scor
plt.fill_between(train_sizes, test_scores_mean - test_scores_std, test_scores_
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training scor
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validati

plt.legend(loc="best")
plt.show()
```



Conclusion

Training Accuracy and Testing Accuracy

1. Training Accuracy (Reg): 0.9795
2. Testing Accuracy (Reg): 0.9641

These accuracy scores indicate how well model performs on the training and testing datasets after regularization. The training accuracy of approximately 0.9795 suggests that the model correctly predicts nearly 97.95% of the training data instances. The testing accuracy of about 0.9641 indicates that the model predicts around 96.41% of the testing data instances correctly. These scores are high, suggesting that the model generalizes well to unseen data.

Classification Reports Training Classification Report (Reg):

1. Precision: Precision measures the accuracy of positive predictions. For class 0 (no stroke), it's 0.97, and for class 1 (stroke), it's 0.99.
2. Recall: Recall measures the proportion of actual positives that were correctly identified by the model. For class 0, it's 0.99, and for class 1, it's 0.97.
3. F1-score: The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It's 0.98 for both classes.
4. Support: The number of samples in each class.

Testing Classification Report (Reg): Similar to the training report, but computed on the testing dataset. Precision, recall, F1-score, and support metrics for class 0 and class 1 are provided. Confusion Matrices Training Confusion Matrix (Reg):

1. True Negatives (TN): 3732 (Correctly predicted no stroke when there was no stroke)
2. False Positives (FP): 55 (Incorrectly predicted stroke when there was no stroke)
3. False Negatives (FN): 100 (Incorrectly predicted no stroke when there was a stroke)
4. True Positives (TP): 3685 (Correctly predicted stroke when there was a stroke)

Testing Confusion Matrix (Reg): Similar to the training confusion matrix, but computed on the testing dataset.

Interpretation

1. Accuracy: Both training and testing accuracies are high, indicating that the model performs well on both the data it was trained on and new, unseen data from the testing set.
2. Precision and Recall: High precision and recall values across both classes (0 and 1) in the classification reports indicate that the model is effective in correctly identifying both strokes and non-strokes.
3. F1-score: The high F1-scores for both classes suggest a good balance between precision and recall, reinforcing the model's ability to perform well across different metrics.
4. Confusion Matrices: The confusion matrices show that the model makes few errors, with a notable number of true positives and true negatives compared to false positives and false negatives.

Conclusion Overall, these metrics indicate that Random Forest model, after regularization, performs robustly and effectively in predicting strokes based on the given dataset. The high accuracy scores, coupled with balanced precision, recall, and F1-scores, suggest that the model is reliable and generalizes well to new data. This regularization has helped mitigate overfitting, resulting in a model that maintains high performance on unseen data while remaining stable during training.

In [ ]: