In [12]:
```
import pandas as pd

# Load the dataset
df = pd.read_csv("medical_insurance.csv")

# Preview the first 5 rows
print("First 5 rows of the dataset:")
display(df.head())

# Shape of the dataset
print(f"\nDataset contains {df.shape[0]} rows and {df.shape[1]} columns.")

# Data type information
print("\nData types and non-null counts:")
display(df.info())
```

```
First 5 rows of the dataset:
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

```
Dataset contains 2772 rows and 7 columns.

Data types and non-null counts:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2772 entries, 0 to 2771
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       2772 non-null   int64  
 1   sex       2772 non-null   object 
 2   bmi       2772 non-null   float64
 3   children  2772 non-null   int64  
 4   smoker    2772 non-null   object 
 5   region    2772 non-null   object 
 6   charges   2772 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 151.7+ KB
None
```

In [14]:
```
# Statistical summary of numerical columns
display(df.describe())

# Unique values in categorical columns
print("\nUnique values in 'sex':", df['sex'].unique())
print("Unique values in 'smoker':", df['smoker'].unique())
print("Unique values in 'region':", df['region'].unique())
```

	age	bmi	children	charges
count	2772.000000	2772.000000	2772.000000	2772.000000
mean	39.109668	30.701349	1.101732	13261.369959
std	14.081459	6.129449	1.214806	12151.768945
min	18.000000	15.960000	0.000000	1121.873900
25%	26.000000	26.220000	0.000000	4687.797000
50%	39.000000	30.447500	1.000000	9333.014350
75%	51.000000	34.770000	2.000000	16577.779500
max	64.000000	53.130000	5.000000	63770.428010

```
Unique values in 'sex': ['female' 'male']
Unique values in 'smoker': ['yes' 'no']
Unique values in 'region': ['southwest' 'southeast' 'northwest' 'northeast']
```

In [16]:
```
# Check for missing values
missing_counts = df.isnull().sum()
print("Missing values per column:")
print(missing_counts)
```

```
Missing values per column:
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

In [18]:
```
# Check for duplicate rows
dup_count = df.duplicated().sum()
print(f"\nNumber of duplicate records: {dup_count}")

# If duplicates exist, drop them
if dup_count > 0:
    df = df.drop_duplicates().reset_index(drop=True)
    print(f"Duplicates dropped. New dataset length: {len(df)}")
```

```
Number of duplicate records: 1435
Duplicates dropped. New dataset length: 1337
```

In [20]:
```
# Define BMI category bins and labels
bins = [0, 18.5, 25, 30, float('inf')]
labels = ['Underweight', 'Normal', 'Overweight', 'Obese']

# Create a new column 'BMI_Category'
df['BMI_Category'] = pd.cut(df['bmi'], bins=bins, labels=labels)

# Check the distribution of the new BMI_Category feature
print("BMI Category counts:")
```

```
print(df['BMI_Category'].value_counts())
```

```
BMI Category counts:
BMI_Category
Obese          704
Overweight     386
Normal         226
Underweight     21
Name: count, dtype: int64
```

In [22]:
```
# Encode binary categorical features
df['sex']    = df['sex'].map({'female': 0, 'male': 1})
df['smoker'] = df['smoker'].map({'no': 0, 'yes': 1})

# One-hot encode multi-category features (region, BMI_Category)
df = pd.get_dummies(df, columns=['region', 'BMI_Category'], drop_first=True)

# Preview the DataFrame after encoding
print("Columns after encoding:")
print(df.columns.tolist())
print("\nFirst 5 rows after encoding:")
display(df.head())
```

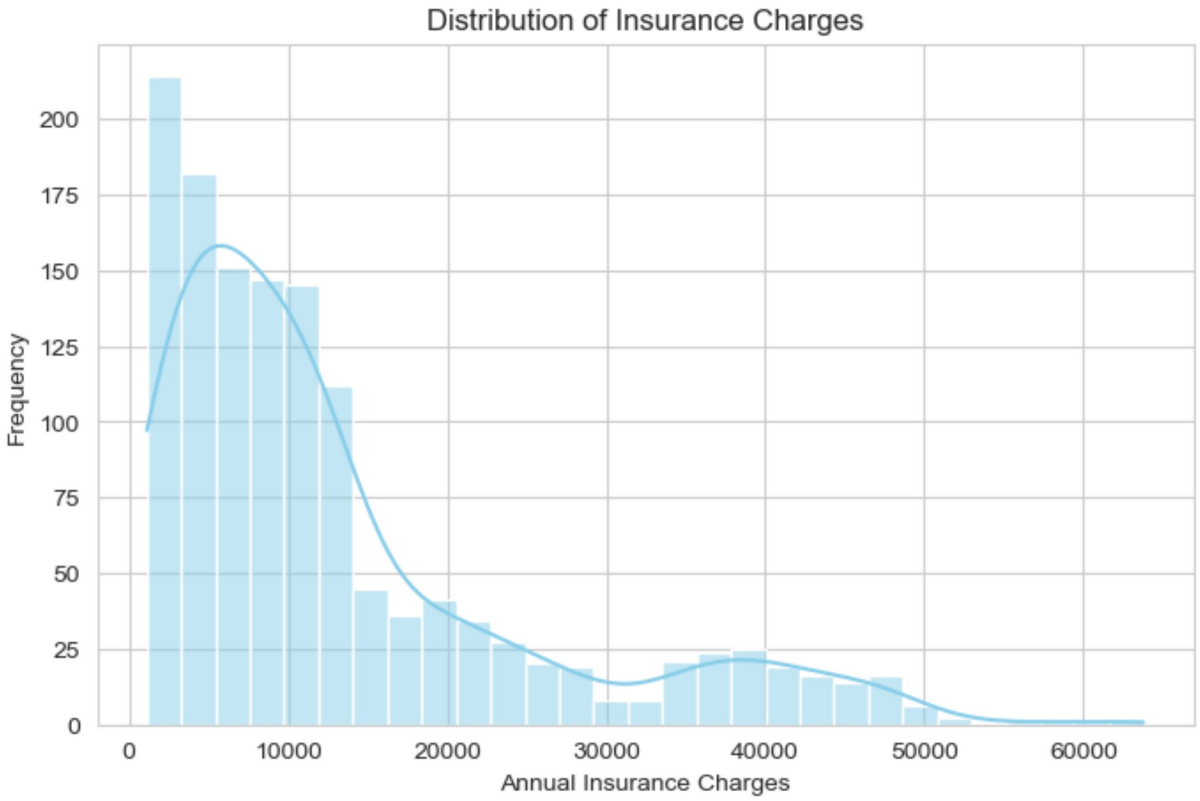
```
Columns after encoding:
['age', 'sex', 'bmi', 'children', 'smoker', 'charges', 'region_northwest', 'region_s
outheast', 'region_southwest', 'BMI_Category_Normal', 'BMI_Category_Overweight', 'BM
I_Category_Obese']

First 5 rows after encoding:
```

	age	sex	bmi	children	smoker	charges	region_northwest	region_southeast	reg
0	19	0	27.900	0	1	16884.92400	False	False	
1	18	1	33.770	1	0	1725.55230	False	True	
2	28	1	33.000	3	0	4449.46200	False	True	
3	33	1	22.705	0	0	21984.47061	True	False	
4	32	1	28.880	0	0	3866.85520	True	False	

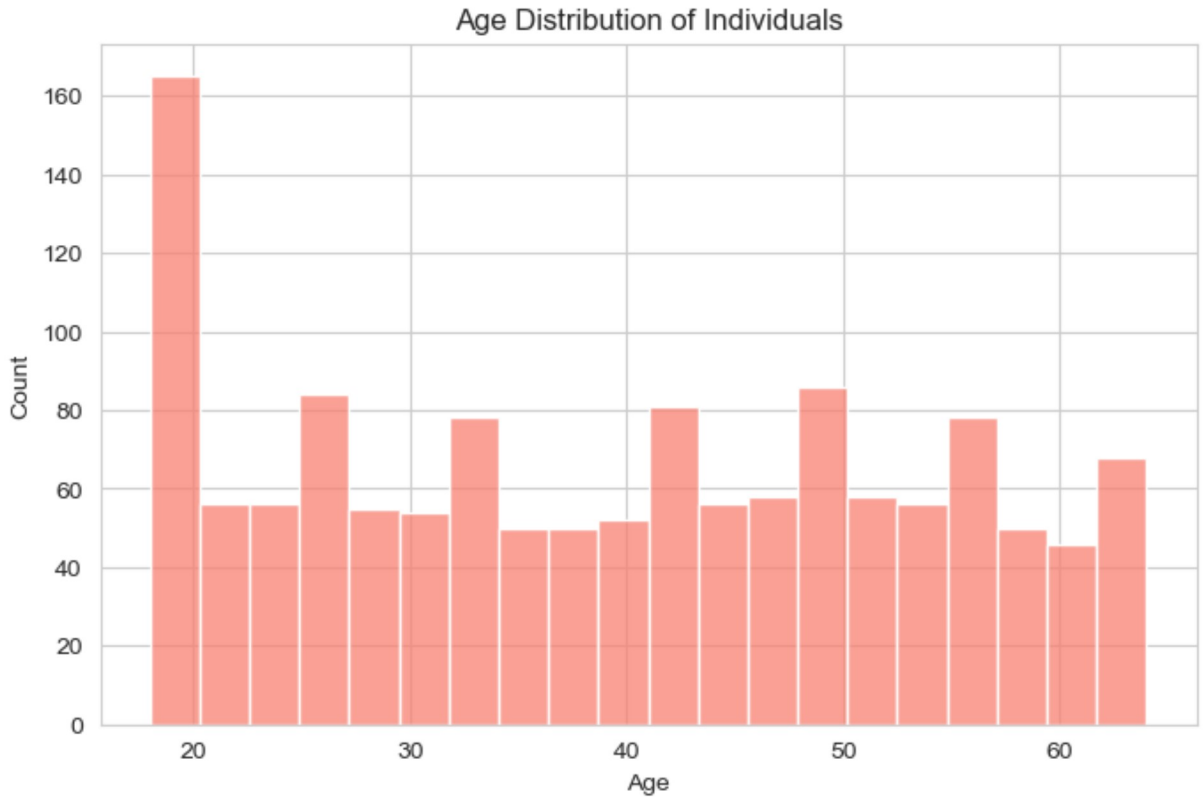
In [24]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a copy of the original data (before encoding) for EDA
df_eda = pd.read_csv("medical_insurance.csv").drop_duplicates()
df_eda['BMI_Category'] = pd.cut(df_eda['bmi'], bins=[0,18.5,25,30,float('inf')], la

# Use seaborn style for better visuals
sns.set_style("whitegrid")
%matplotlib inline
```

In [26]:
```
plt.figure(figsize=(8,5))
sns.histplot(df_eda['charges'], kde=True, color='skyblue')
plt.title('Distribution of Insurance Charges')
plt.xlabel('Annual Insurance Charges')
```

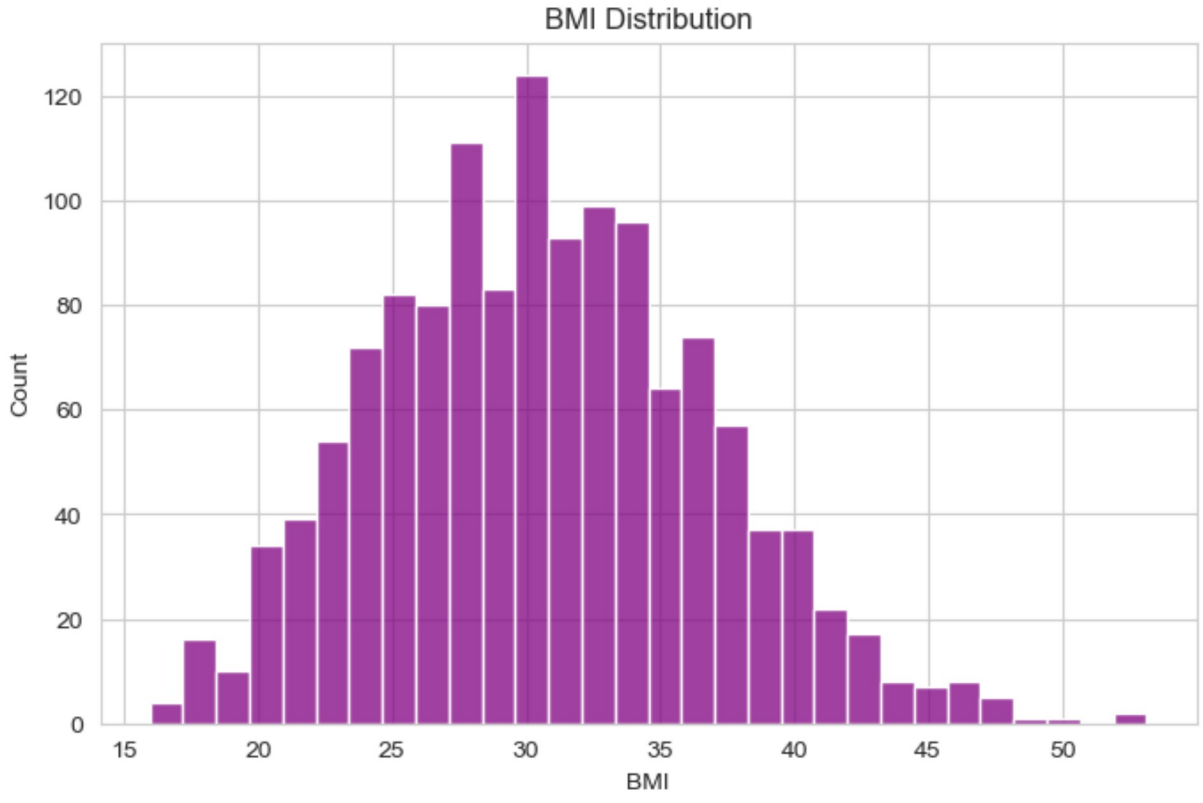
```
plt.ylabel('Frequency')
plt.show()
```



In [28]:
```
plt.figure(figsize=(8,5))
sns.histplot(df_eda['age'], bins=20, color='salmon')
plt.title('Age Distribution of Individuals')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

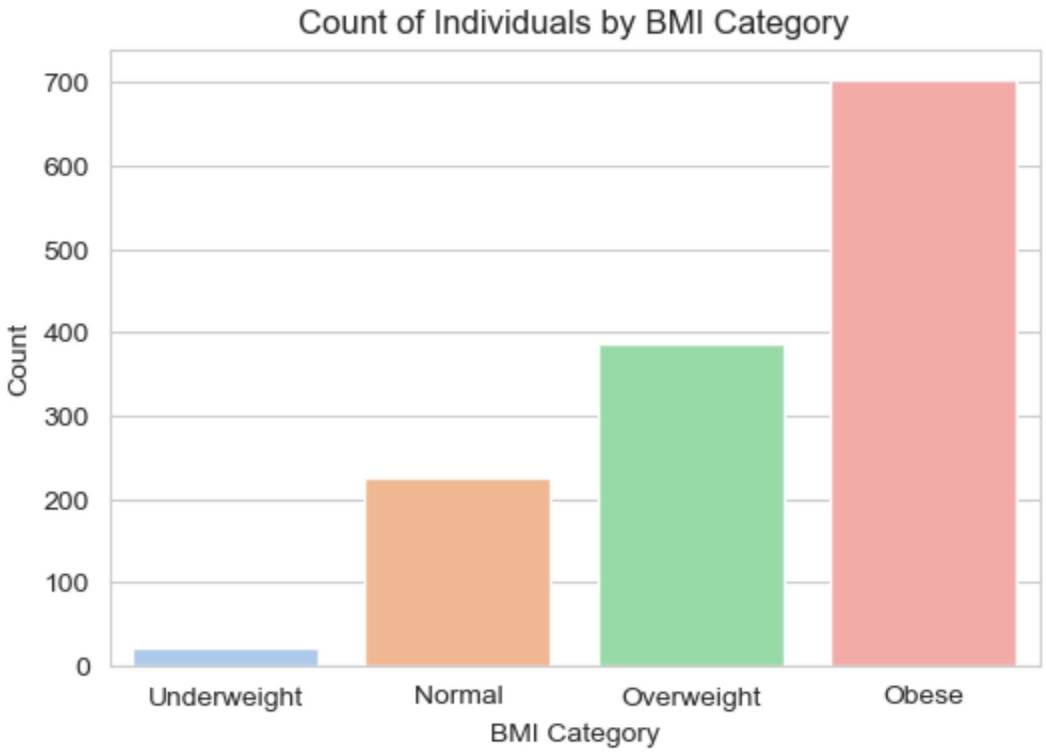


In [30]:
```
plt.figure(figsize=(8,5))
sns.histplot(df_eda['bmi'], bins=30, color='purple')
plt.title('BMI Distribution')
plt.xlabel('BMI')
plt.ylabel('Count')
plt.show()
```



In [32]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='BMI_Category', data=df_eda, palette='pastel',
              order=['Underweight','Normal','Overweight','Obese'])
plt.title('Count of Individuals by BMI Category')
plt.xlabel('BMI Category')
plt.ylabel('Count')
plt.show()
```

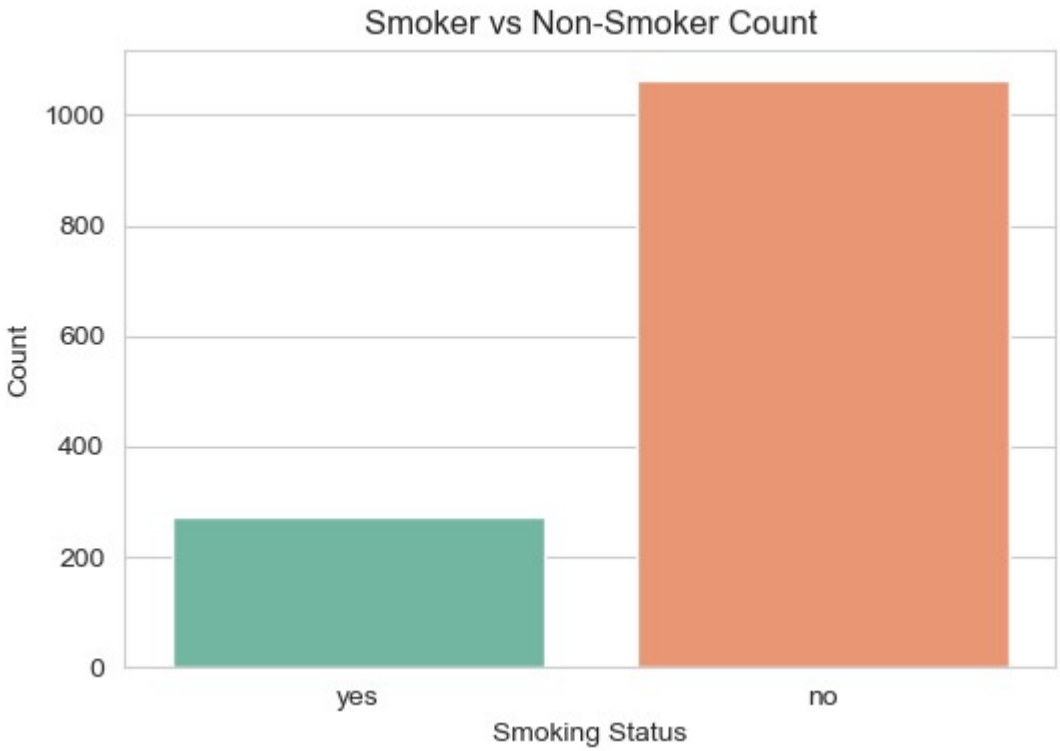
```
C:\Users\naruk\AppData\Local\Temp\ipykernel_23348\3579692938.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14
.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='BMI_Category', data=df_eda, palette='pastel',
```



In [34]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='smoker', data=df_eda, palette='Set2')
plt.title('Smoker vs Non-Smoker Count')
plt.xlabel('Smoking Status')
plt.ylabel('Count')
plt.show()

plt.figure(figsize=(6,4))
sns.countplot(x='sex', data=df_eda, palette='Set2')
plt.title('Gender Distribution')
plt.xlabel('Sex')
plt.ylabel('Count')
plt.show()

plt.figure(figsize=(6,4))
sns.countplot(x='region', data=df_eda, palette='Set3')
plt.title('Count of Individuals by Region')
plt.xlabel('Region')
```

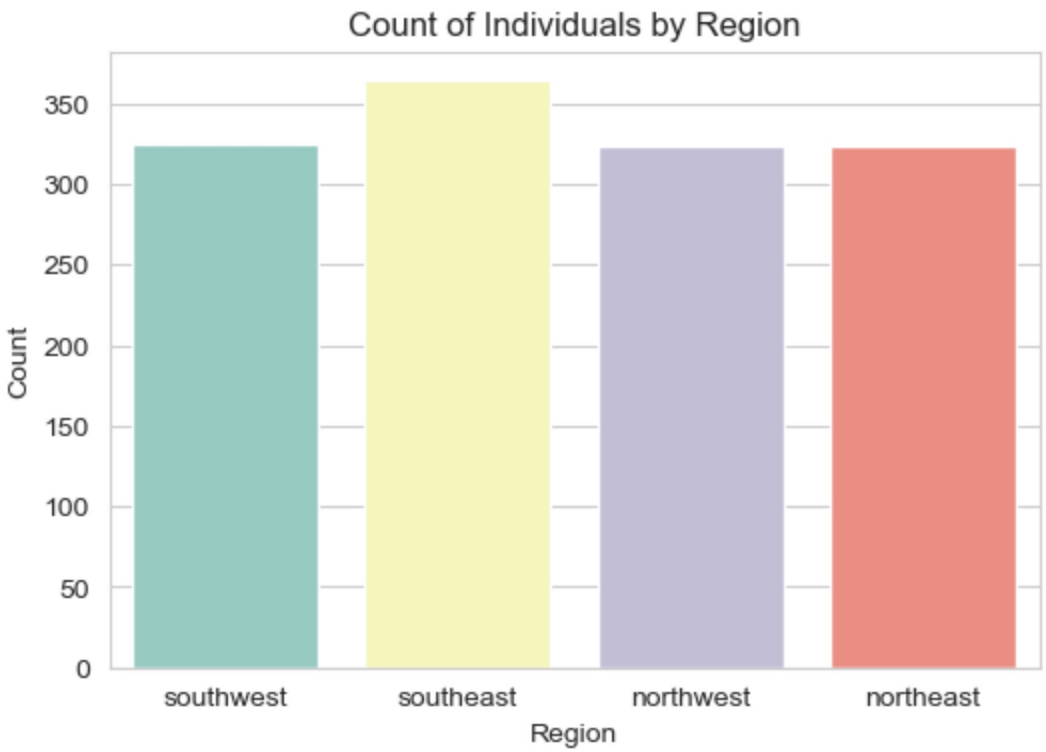
```
plt.ylabel('Count')
plt.show()
```

```
C:\Users\naruk\AppData\Local\Temp\ipykernel_23348\2060321097.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14
.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='smoker', data=df_eda, palette='Set2')
```



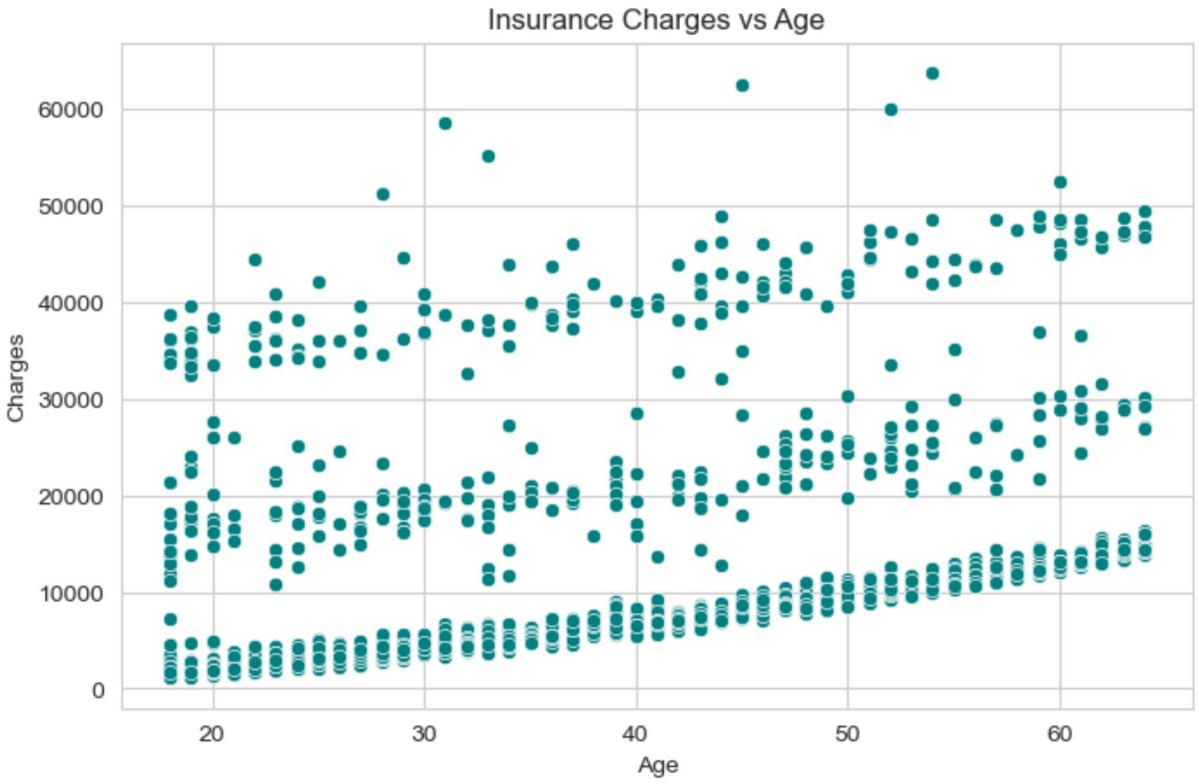

```
C:\Users\naruk\AppData\Local\Temp\ipykernel_23348\2060321097.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14
.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='sex', data=df_eda, palette='Set2')
```


```
C:\Users\naruk\AppData\Local\Temp\ipykernel_23348\2060321097.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14
.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='region', data=df_eda, palette='Set3')
```

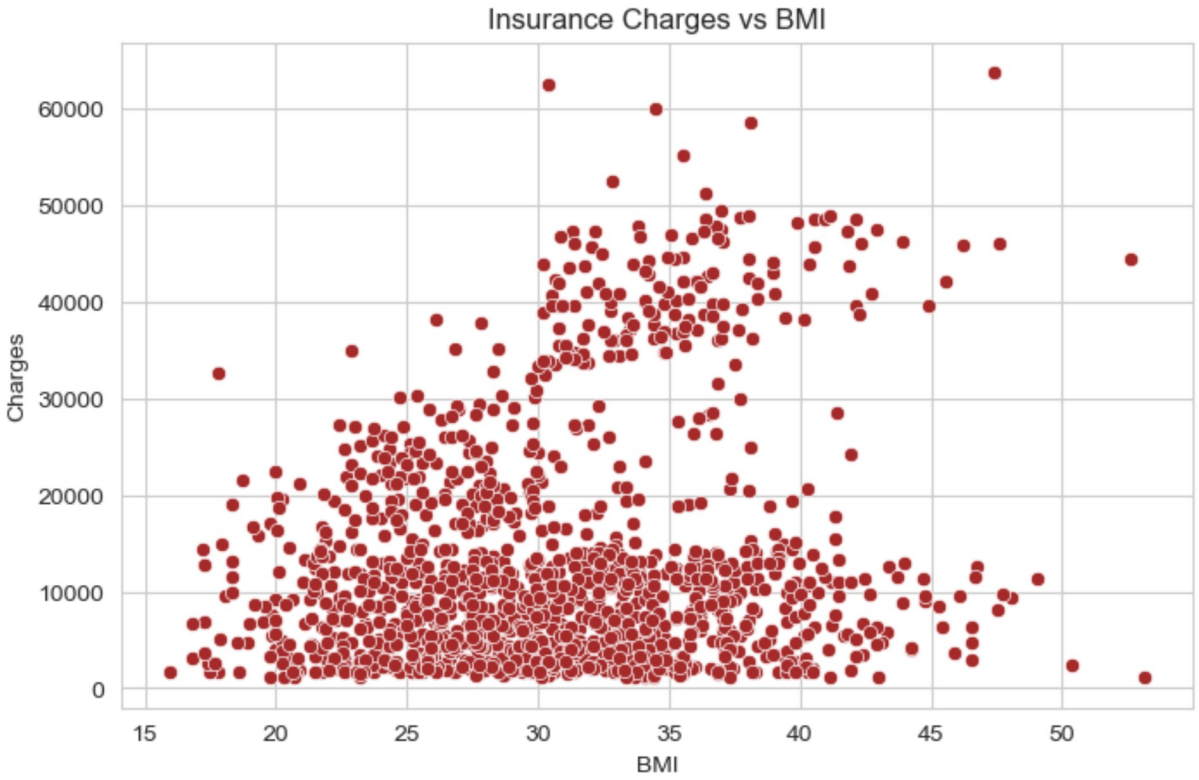


In [36]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(x='age', y='charges', data=df_eda, color='teal')
plt.title('Insurance Charges vs Age')
```

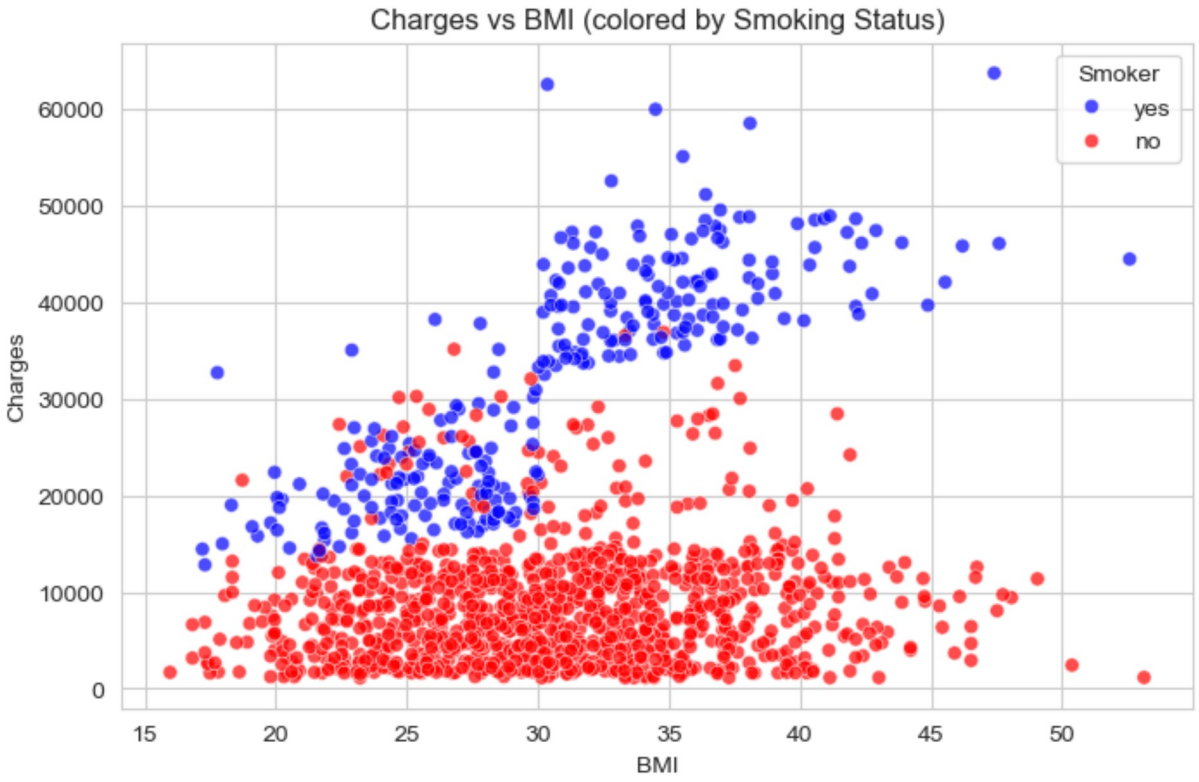
```
plt.xlabel('Age')
plt.ylabel('Charges')
plt.show()
```



In [38]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(x='bmi', y='charges', data=df_eda, color='brown')
plt.title('Insurance Charges vs BMI')
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.show()
```

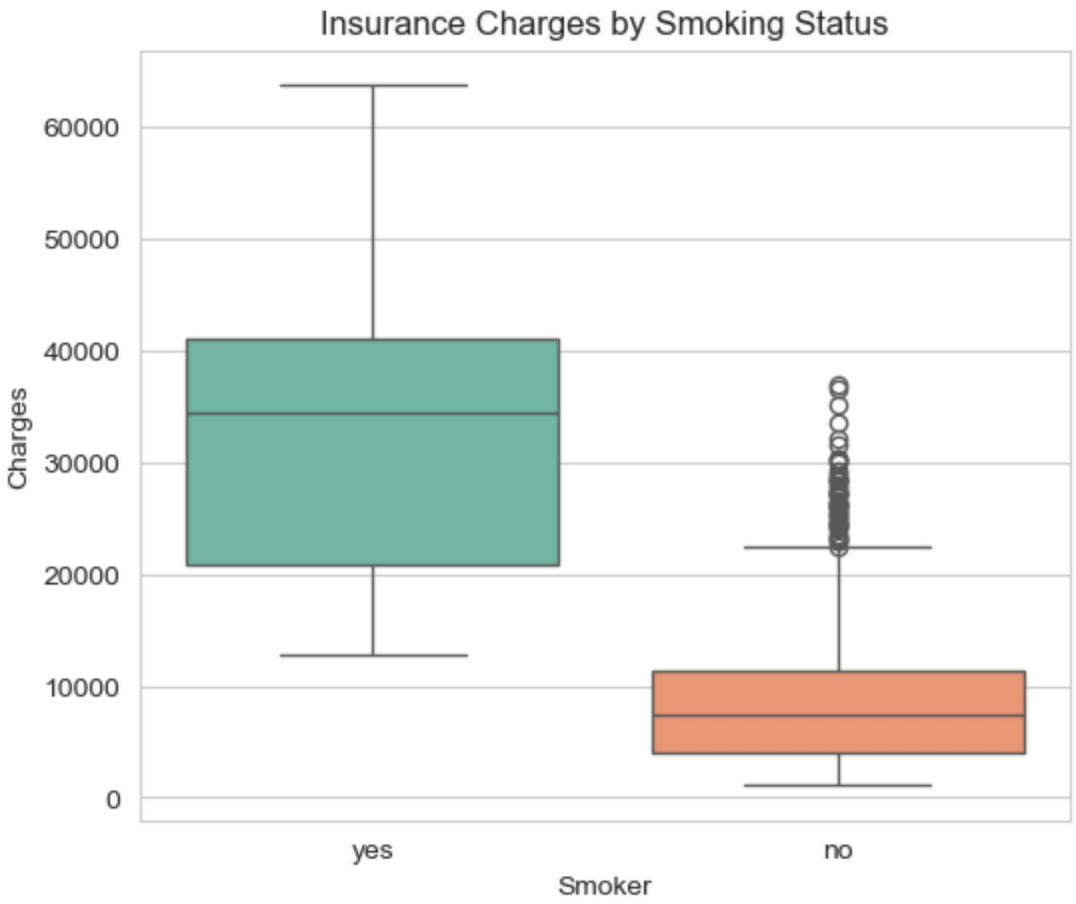


In [40]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(x='bmi', y='charges', hue='smoker', data=df_eda, palette=['blue','r
plt.title('Charges vs BMI (colored by Smoking Status)')
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.legend(title='Smoker')
plt.show()
```



In [42]:
```
plt.figure(figsize=(6,5))
sns.boxplot(x='smoker', y='charges', data=df_eda, palette='Set2')
plt.title('Insurance Charges by Smoking Status')
plt.xlabel('Smoker')
plt.ylabel('Charges')
plt.show()
```

```
C:\Users\naruk\AppData\Local\Temp\ipykernel_23348\3527151539.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14
.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='smoker', y='charges', data=df_eda, palette='Set2')
```

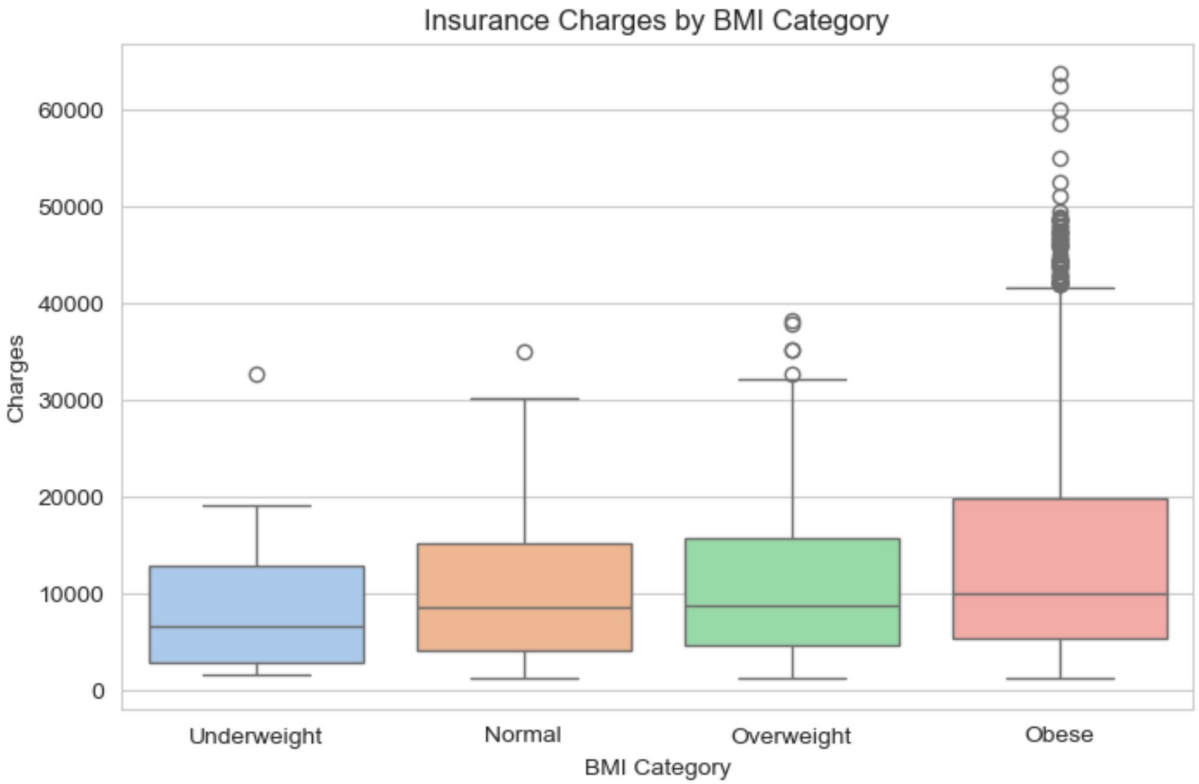


In [44]:
```
mean_charges = df_eda.groupby('smoker')['charges'].mean()
print("Average charges for non-smokers:", mean_charges['no'])
print("Average charges for smokers:", mean_charges['yes'])
```

```
Average charges for non-smokers: 8440.660306508938
Average charges for smokers: 32050.23183153285
```

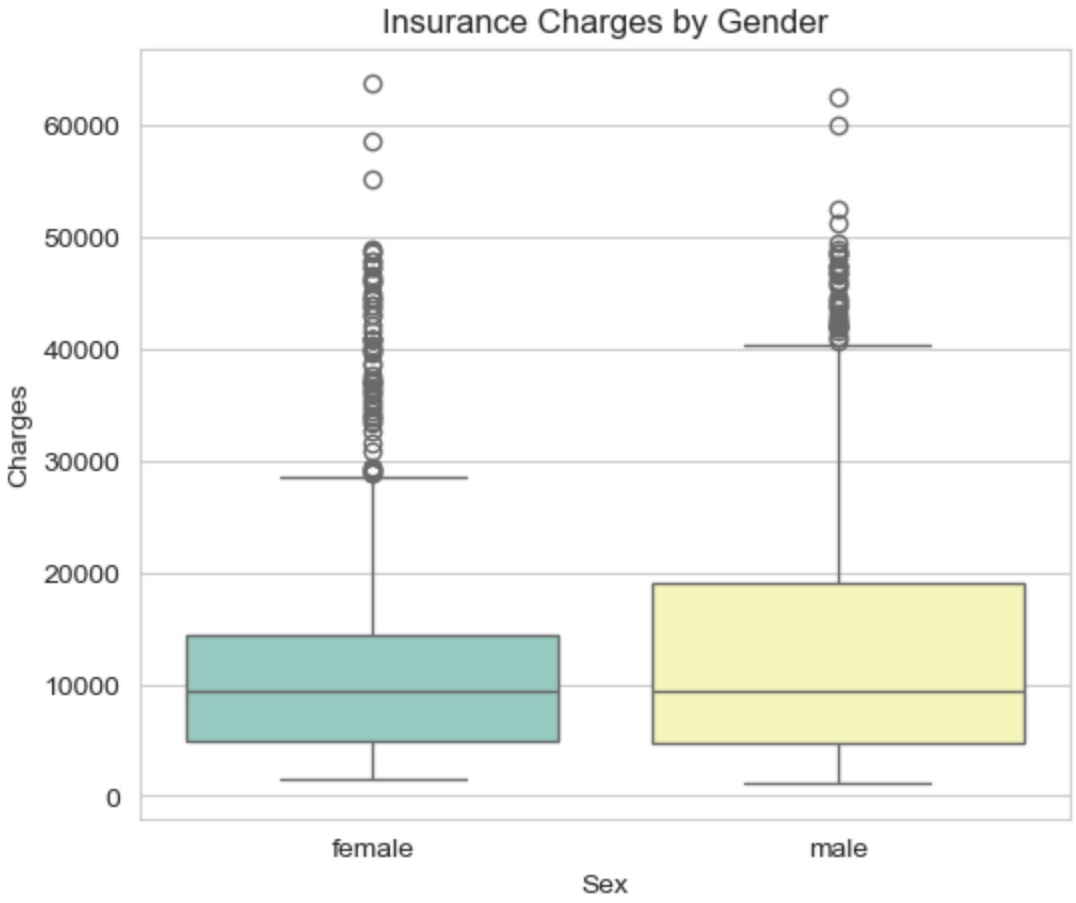
In [46]:
```
plt.figure(figsize=(8,5))
sns.boxplot(x='BMI_Category', y='charges', data=df_eda, order=['Underweight','Norma
plt.title('Insurance Charges by BMI Category')
plt.xlabel('BMI Category')
plt.ylabel('Charges')
plt.show()
```

```
C:\Users\naruk\AppData\Local\Temp\ipykernel_23348\3315977569.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14
.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='BMI_Category', y='charges', data=df_eda, order=['Underweight','Norm
al','Overweight','Obese'], palette='pastel')
```



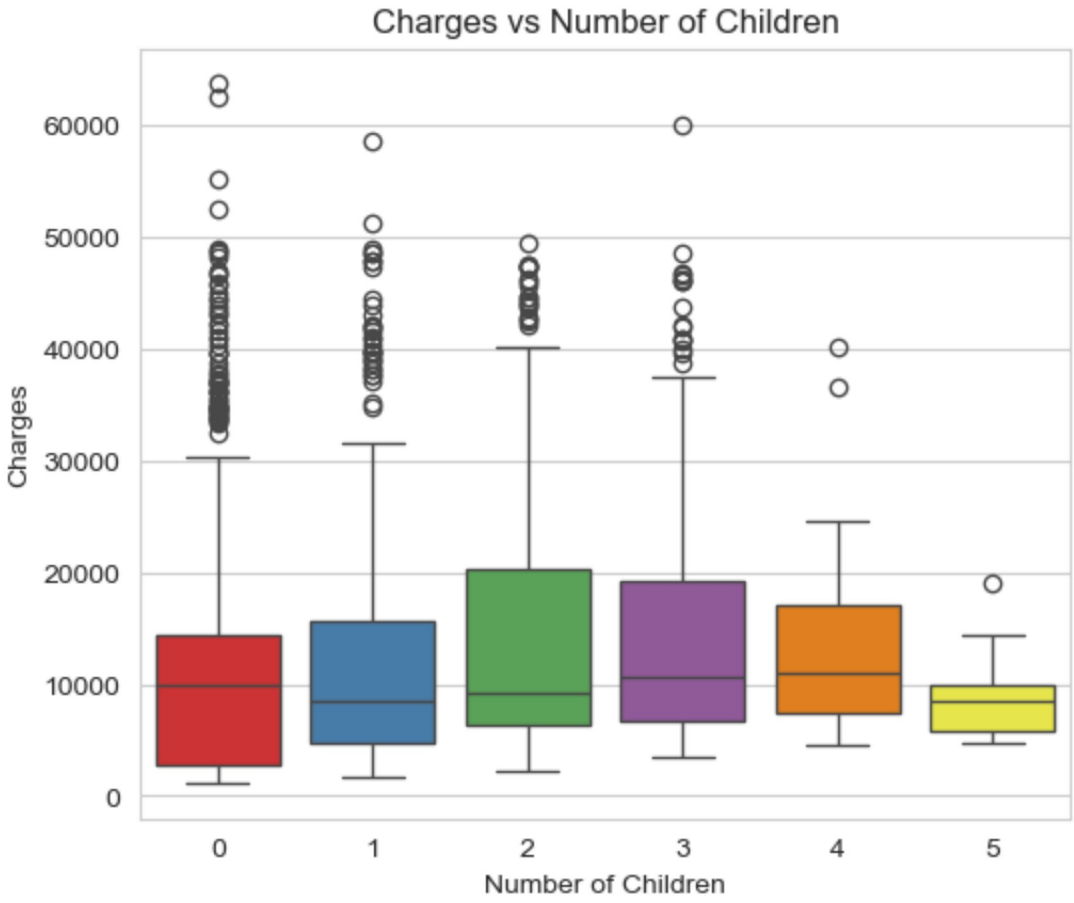
In [48]:
```
plt.figure(figsize=(6,5))
sns.boxplot(x='sex', y='charges', data=df_eda, palette='Set3')
plt.title('Insurance Charges by Gender')
plt.xlabel('Sex')
plt.ylabel('Charges')
plt.show()
```

```
C:\Users\naruk\AppData\Local\Temp\ipykernel_23348\937498998.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14
.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='sex', y='charges', data=df_eda, palette='Set3')
```



In [50]:
```
plt.figure(figsize=(6,5))
sns.boxplot(x='children', y='charges', data=df_eda, palette='Set1')
plt.title('Charges vs Number of Children')
plt.xlabel('Number of Children')
plt.ylabel('Charges')
plt.show()
```

```
C:\Users\naruk\AppData\Local\Temp\ipykernel_23348\180229155.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14
.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='children', y='charges', data=df_eda, palette='Set1')
```



In [52]:
```
# Create age group categories for analysis
df_eda['age_group'] = pd.cut(df_eda['age'], bins=[17,30,50,100], labels=['<=30','31

# Calculate average charges for smokers vs non-smokers in each age group
group_charges = df_eda.groupby(['age_group','smoker'])['charges'].mean().reset_inde
print(group_charges.pivot(index='age_group', columns='smoker', values='charges'))
```

```
smoker               no           yes
age_group
<=30        4470.419842  27528.078343
31-50       8186.653963  32018.276539
>50        13540.277993  38820.223082
```

```
C:\Users\naruk\AppData\Local\Temp\ipykernel_23348\3039530814.py:5: FutureWarning: Th
e default of observed=False is deprecated and will be changed to True in a future ve
rsion of pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
  group_charges = df_eda.groupby(['age_group','smoker'])['charges'].mean().reset_ind
ex()
```

In [54]:
```
# Define obese smokers and non-obese non-smokers groups
obese_smokers = df_eda[(df_eda['BMI_Category']=='Obese') & (df_eda['smoker']=='yes'
nonobese_nonsmokers = df_eda[(df_eda['BMI_Category']!='Obese') & (df_eda['smoker']=

print("Average charges (Obese Smokers):", obese_smokers['charges'].mean())
print("Average charges (Non-obese Non-smokers):", nonobese_nonsmokers['charges'].me
```

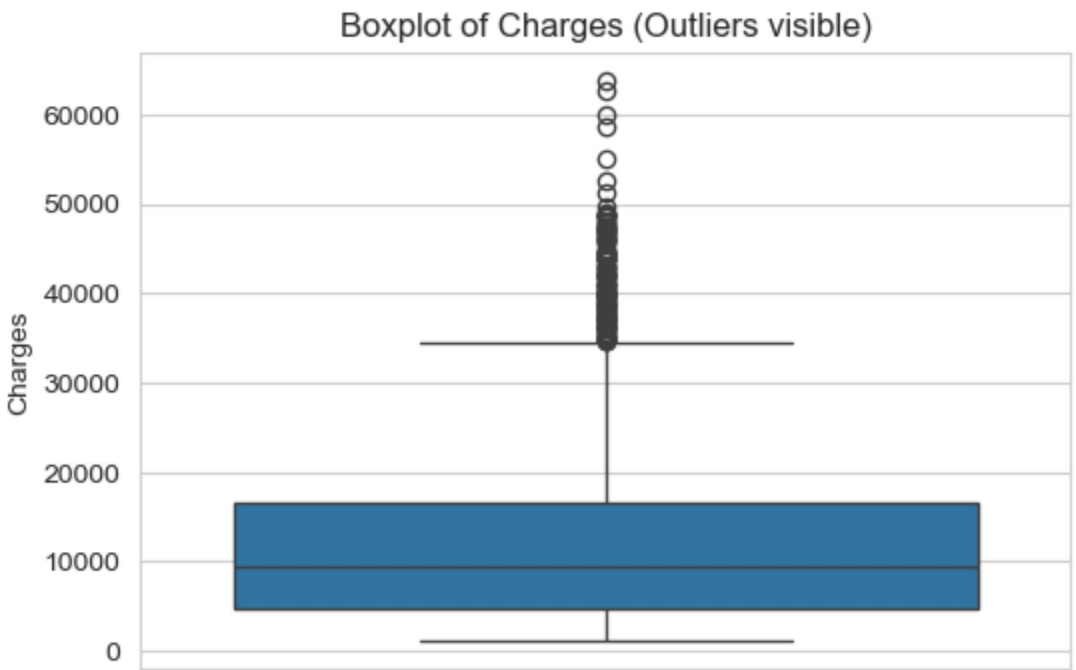
```
Average charges (Obese Smokers): 41692.80899152778
Average charges (Non-obese Non-smokers): 7966.944149520875
```

In [56]:
```
# Top 5 highest charges
top_charges = df_eda.nlargest(5, 'charges')[['age','sex','bmi','children','smoker',
print("Top 5 highest charges in the dataset:")
display(top_charges)
```

```
Top 5 highest charges in the dataset:
```

	age	sex	bmi	children	smoker	region	charges
543	54	female	47.410	0	yes	southeast	63770.42801
1300	45	male	30.360	0	yes	southeast	62592.87309
1230	52	male	34.485	3	yes	northwest	60021.39897
577	31	female	38.095	1	yes	northeast	58571.07448
819	33	female	35.530	0	yes	northwest	55135.40209

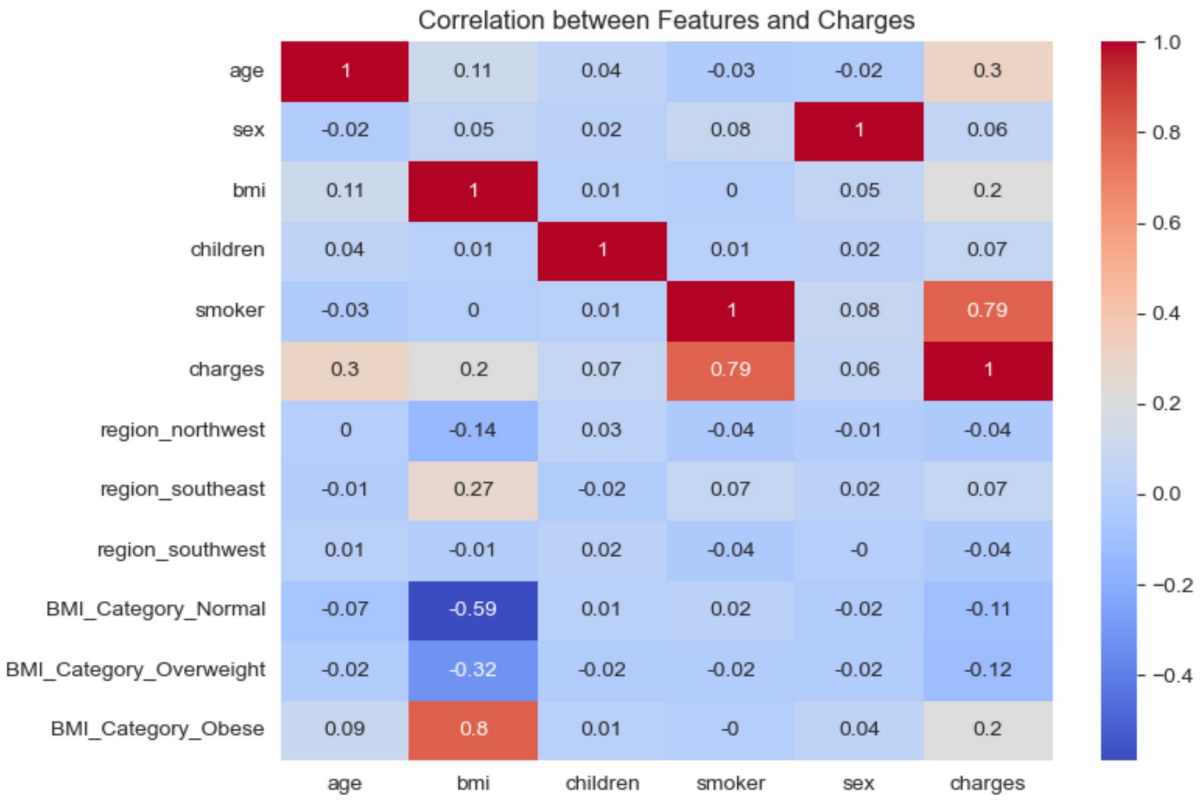
In [58]:
```
plt.figure(figsize=(6,4))
sns.boxplot(y='charges', data=df_eda)
plt.title('Boxplot of Charges (Outliers visible)')
plt.ylabel('Charges')
plt.show()
```



In [60]:
```
# Compute correlation matrix on the encoded data
corr_matrix = df.corr(numeric_only=True)

# Display correlation of features with the target 'charges'
print("Correlation of features with charges:")
print(corr_matrix['charges'].sort_values(ascending=False))
```

```
Correlation of features with charges:
charges                    1.000000
smoker                     0.787234
age                        0.298308
BMI_Category_Obese         0.200501
bmi                        0.198401
region_southeast           0.073578
children                   0.067389
sex                        0.058044
region_northwest          -0.038695
region_southwest          -0.043637
BMI_Category_Normal       -0.105946
BMI_Category_Overweight   -0.120059
Name: charges, dtype: float64
```

In [62]:
```
plt.figure(figsize=(8,6))
sns.heatmap(corr_matrix[['age','bmi','children','smoker','sex','charges']].round(2)
plt.title('Correlation between Features and Charges')
plt.show()
```



In [71]:
```
!pip install xgboost

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


# Prepare feature matrix X and target vector y
X = df.drop('charges', axis=1)
```

```
y = df['charges']

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Define the models with default hyperparameters
models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(random_state=42),
    "Lasso Regression": Lasso(random_state=42),
    "Random Forest": RandomForestRegressor(random_state=42),
    "XGBoost": XGBRegressor(random_state=42, use_label_encoder=False, eval_metric='
}

# Train each model and evaluate on test set
results = []  # to store performance metrics for each model
for name, model in models.items():
    model.fit(X_train, y_train)               # train the model
    y_pred = model.predict(X_test)            # make predictions on test set
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    mae  = mean_absolute_error(y_test, y_pred)
    r2   = r2_score(y_test, y_pred)
    results.append({
        'Model': name,
        'RMSE': rmse,
        'MAE': mae,
        'R2': r2
    })

# Convert results to a DataFrame for a neat display
results_df = pd.DataFrame(results)
results_df = results_df.sort_values(by='R2', ascending=False)
display(results_df)
```

```
Requirement already satisfied: xgboost in c:\users\naruk\anaconda3\lib\site-packages
(3.0.2)
Requirement already satisfied: numpy in c:\users\naruk\anaconda3\lib\site-packages
(from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\naruk\anaconda3\lib\site-packages
(from xgboost) (1.13.1)
```

```
C:\Users\naruk\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: Futur
eWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calc
ulate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
C:\Users\naruk\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: Futur
eWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calc
ulate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
C:\Users\naruk\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: Futur
eWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calc
ulate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
C:\Users\naruk\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: Futur
eWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calc
ulate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
C:\Users\naruk\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [1
9:14:24] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
C:\Users\naruk\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: Futur
eWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calc
ulate the root mean squared error, use the function'root_mean_squared_error'.
  warnings.warn(
```

	Model	RMSE	MAE	R2
3	Random Forest	4687.357764	2641.434302	0.880432
4	XGBoost	5105.708162	2887.280268	0.858137
2	Lasso Regression	6019.393890	4334.011290	0.802820
1	Ridge Regression	6023.350575	4332.246193	0.802560
0	Linear Regression	6024.004673	4338.923817	0.802517

In [72]:

```
import mlflow
import mlflow.sklearn

# Set an experiment name (creates a new experiment if not exists)
mlflow.set_experiment("medical-insurance-cost-prediction")

best_model_name = None
best_r2 = -1
best_run_id = None

for result in results:
    model_name = result['Model']
    rmse = result['RMSE']
    mae = result['MAE']
    r2 = result['R2']
    model_obj = models[model_name]  # get the trained model object from our diction

    # Start a new MLflow run for this model
```

```
    with mlflow.start_run(run_name=model_name):
        # Log model parameters (if any hyperparameters relevant)
        # Here we log the model name and for regularized models, the alpha if appli
        mlflow.log_param("model_name", model_name)
        if model_name == "Ridge Regression":
            mlflow.log_param("alpha", model_obj.alpha)
        if model_name == "Lasso Regression":
            mlflow.log_param("alpha", model_obj.alpha)
        if model_name == "Random Forest":
            mlflow.log_param("n_estimators", model_obj.n_estimators)
        if model_name == "XGBoost":
            mlflow.log_param("n_estimators", model_obj.n_estimators)

        # Log metrics
        mlflow.log_metric("RMSE", rmse)
        mlflow.log_metric("MAE", mae)
        mlflow.log_metric("R2", r2)

        # Log the model artifact (save the model in MLflow)
        mlflow.sklearn.log_model(model_obj, artifact_path="model")

        # If this model is the best so far, record its run ID and name
        if r2 > best_r2:
            best_r2 = r2
            best_model_name = model_name
            best_run_id = mlflow.active_run().info.run_id  # capture the run ID of

# After logging all models, print out the best model info
print(f"Best model: {best_model_name} with R2 = {best_r2:.3f}")
print(f"MLflow run ID for best model: {best_run_id}")
```

```
2025/07/22 19:14:24 WARNING mlflow.utils.git_utils: Failed to import Git (the Git ex
ecutable is probably not on your PATH), so Git SHA is not available. Error: Failed t
o initialize: Bad git executable.
The git executable must be specified in one of the following ways:
    - be included in your $PATH
    - be set via $GIT_PYTHON_GIT_EXECUTABLE
    - explicitly set via git.refresh(<full-path-to-git-executable>)

All git commands will error until this is rectified.

This initial message can be silenced or aggravated in the future by setting the
$GIT_PYTHON_REFRESH environment variable. Use one of the following values:
    - quiet|q|silence|s|silent|none|n|0: for no message or exception
    - warn|w|warning|log|l|1: for a warning message (logging level CRITICAL, display
ed by default)
    - error|e|exception|raise|r|2: for a raised exception

Example:
    export GIT_PYTHON_REFRESH=quiet

2025/07/22 19:14:24 WARNING mlflow.models.model: `artifact_path` is deprecated. Plea
se use `name` instead.
2025/07/22 19:14:30 WARNING mlflow.models.model: Model logged without a signature an
d input example. Please set `input_example` parameter when logging the model to auto
infer the model signature.
2025/07/22 19:14:30 WARNING mlflow.models.model: `artifact_path` is deprecated. Plea
se use `name` instead.
2025/07/22 19:14:33 WARNING mlflow.models.model: Model logged without a signature an
d input example. Please set `input_example` parameter when logging the model to auto
infer the model signature.
2025/07/22 19:14:33 WARNING mlflow.models.model: `artifact_path` is deprecated. Plea
se use `name` instead.
2025/07/22 19:14:36 WARNING mlflow.models.model: Model logged without a signature an
d input example. Please set `input_example` parameter when logging the model to auto
infer the model signature.
2025/07/22 19:14:36 WARNING mlflow.models.model: `artifact_path` is deprecated. Plea
se use `name` instead.
2025/07/22 19:14:39 WARNING mlflow.models.model: Model logged without a signature an
d input example. Please set `input_example` parameter when logging the model to auto
infer the model signature.
2025/07/22 19:14:39 WARNING mlflow.models.model: `artifact_path` is deprecated. Plea
se use `name` instead.
2025/07/22 19:14:42 WARNING mlflow.models.model: Model logged without a signature an
d input example. Please set `input_example` parameter when logging the model to auto
infer the model signature.
Best model: Random Forest with R2 = 0.880
MLflow run ID for best model: 2eb9a9c909324c0180b160de504b4585
```

In [73]:
```
if best_run_id:
    model_registry_name = "InsuranceChargesBestModel"
    try:
        mlflow.register_model(model_uri=f"runs:/{best_run_id}/model", name=model_re
        print(f"Model registered as '{model_registry_name}' in MLflow Model Registr
    except Exception as e:
        print("Model registration failed (perhaps MLflow tracking server is not con
        print(e)
```

```
Successfully registered model 'InsuranceChargesBestModel'.
2025/07/22 19:14:42 WARNING mlflow.tracking._model_registry.fluent: Run with id 2eb9
a9c909324c0180b160de504b4585 has no artifacts at artifact path 'model', registering
model based on models:/m-9289e318043d43058d962cd8b44a69cb instead
Model registered as 'InsuranceChargesBestModel' in MLflow Model Registry.
Created version '1' of model 'InsuranceChargesBestModel'.
```

In [80]:
```
import pickle

best_model = models[best_model_name]  # retrieve the best model object
# Save the model to a file
with open("best_model.pkl", "wb") as f:
    pickle.dump(best_model, f)

print(f"Best model ({best_model_name}) saved to best_model.pkl")
```

```
Best model (Random Forest) saved to best_model.pkl
```

In [ ]:
```
```

In [46]:
```
# ------------------------------------------------------------
# Medical Insurance Charge Prediction - Streamlit App
# Trains model from scratch on each rerun (no .pkl needed)
# ------------------------------------------------------------

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import streamlit as st

# Try XGBoost; fall back to RandomForest if not installed
try:
    from xgboost import XGBRegressor
    USE_XGB = True
except Exception:
    from sklearn.ensemble import RandomForestRegressor
    USE_XGB = False

# ----------------------------
# 1. Load Data (cached)
# ----------------------------
@st.cache_data
def load_data(path: str = "medical_insurance.csv") -> pd.DataFrame:
```

```
    df = pd.read_csv(path)
    df = df.drop_duplicates().reset_index(drop=True)
    return df

df_raw = load_data()

# ----------------------------
# 2. Preprocess Data
# ----------------------------
def preprocess(df: pd.DataFrame) -> pd.DataFrame:
    """Return encoded dataframe ready for modeling."""
    df = df.copy()

    # BMI Category
    bins = [0, 18.5, 25, 30, float("inf")]
    labels = ["Underweight", "Normal", "Overweight", "Obese"]
    df["BMI_Category"] = pd.cut(df["bmi"], bins=bins, labels=labels, include_lowest

    # Binary encodes
    df["sex"] = df["sex"].map({"female": 0, "male": 1})
    df["smoker"] = df["smoker"].map({"no": 0, "yes": 1})

    # Dummies
    df_enc = pd.get_dummies(df, columns=["region", "BMI_Category"], drop_first=Fals

    return df_enc

df_model = preprocess(df_raw)

# Separate features / target
X = df_model.drop("charges", axis=1)
y = df_model["charges"]

# ----------------------------
# 3. Train Model (each rerun)
# ----------------------------
with st.spinner("Training model..."):
    if USE_XGB:
        model = XGBRegressor(
            n_estimators=300,
            learning_rate=0.05,
            max_depth=4,
            subsample=0.9,
            colsample_bytree=0.9,
            random_state=42,
            use_label_encoder=False,
            eval_metric="rmse",
        )
    else:
        # Fallback model
        from sklearn.ensemble import RandomForestRegressor
        model = RandomForestRegressor(
            n_estimators=300,
            random_state=42,
            n_jobs=-1,
        )
```

```
    model.fit(X, y)

st.success("Model trained successfully!")

# ----------------------------
# Sidebar Inputs
# ----------------------------
st.sidebar.header("Input Patient Data")

age = st.sidebar.slider(
    "Age",
    min_value=int(df_raw["age"].min()),
    max_value=int(df_raw["age"].max()),
    value=30,
)

sex_option = st.sidebar.selectbox("Sex", ["Female", "Male"])
bmi = st.sidebar.slider(
    "BMI",
    min_value=float(df_raw["bmi"].min()),
    max_value=float(df_raw["bmi"].max()),
    value=25.0,
)
children = st.sidebar.number_input(
    "Number of Children", min_value=0, max_value=int(df_raw["children"].max()), val
)
smoker_option = st.sidebar.selectbox("Smoker", ["No", "Yes"])
region_option = st.sidebar.selectbox(
    "Region", [r.capitalize() for r in sorted(df_raw["region"].unique())]
)

# ----------------------------
# 4. Encode User Input
# ----------------------------
def bmi_to_cat(val: float) -> str:
    if val < 18.5:
        return "Underweight"
    elif val < 25:
        return "Normal"
    elif val < 30:
        return "Overweight"
    else:
        return "Obese"

sex_val = 1 if sex_option == "Male" else 0
smoker_val = 1 if smoker_option == "Yes" else 0
region_val = region_option.lower()
bmi_cat = bmi_to_cat(bmi)

# Build input row with *all* model columns
input_dict = {c: 0 for c in X.columns}
input_dict["age"] = age
input_dict["sex"] = sex_val
input_dict["bmi"] = bmi
input_dict["children"] = children
```

```
input_dict["smoker"] = smoker_val

region_col = f"region_{region_val}"
bmi_col = f"BMI_Category_{bmi_cat}"
if region_col in input_dict:
    input_dict[region_col] = 1
if bmi_col in input_dict:
    input_dict[bmi_col] = 1

input_df = pd.DataFrame([input_dict], columns=X.columns)

# ----------------------------
# 5. Predict
# ----------------------------
pred_charge = float(model.predict(input_df)[0])

st.subheader("Predicted Annual Insurance Charge")
st.metric(label="USD", value=f"${pred_charge:,.2f}")

# ----------------------------
# 6. Optional EDA
# ----------------------------
st.markdown("---")
st.subheader("Explore the Data")

if st.checkbox("Show data sample"):
    st.dataframe(df_raw.head())

if st.checkbox("Distribution of Insurance Charges"):
    fig, ax = plt.subplots(figsize=(6, 4))
    sns.histplot(df_raw["charges"], kde=True, ax=ax, color="teal")
    ax.set_xlabel("Charges")
    st.pyplot(fig)

if st.checkbox("Charges: Smokers vs Non-Smokers"):
    fig, ax = plt.subplots(figsize=(6, 4))
    sns.boxplot(x="smoker", y="charges", data=df_raw, ax=ax, palette="Set2")
    ax.set_xlabel("Smoker (no/yes)")
    st.pyplot(fig)

if st.checkbox("BMI vs Charges (colored by smoker)"):
    fig, ax = plt.subplots(figsize=(6, 4))
    sns.scatterplot(
        x="bmi", y="charges", hue="smoker", data=df_raw, palette="Set1", ax=ax, alp
    )
    ax.set_xlabel("BMI")
    st.pyplot(fig)

if st.checkbox("Correlation Heatmap"):
    fig, ax = plt.subplots(figsize=(8, 5))
    corr = df_model.corr(numeric_only=True)
    sns.heatmap(corr, cmap="coolwarm", center=0, ax=ax)
    st.pyplot(fig)

st.markdown("----")
st.caption(
```

```
    "Model retrains on each rerun. To improve speed for repeated interaction, "
    "you can cache the trained model using `@st.cache_resource`."
)
```

```
2025-07-22 20:20:17.313 No runtime found, using MemoryCacheStorageManager
C:\Users\naruk\anaconda3\Lib\site-packages\xgboost\training.py:183: UserWarning: [2
0:20:17] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

Out[46]: DeltaGenerator()

In [ ]:
```
!streamlit run P3fstlit.py
```

Old COde

In [ ]:
```
# import warnings
# warnings.filterwarnings("ignore", category=FutureWarning)  # Ignore harmless futu

# import pandas as pd
# import numpy as np
# import seaborn as sns
# import matplotlib.pyplot as plt
# import streamlit as st
# from xgboost import XGBRegressor
# # (If preferred, you can use RandomForestRegressor from sklearn instead of XGBReg

# # Set seaborn style for better visuals
# sns.set_style("darkgrid")

# # Title and description for the app
# st.title("Medical Insurance Charge Prediction")
# st.write("This app predicts **insurance charges** based on user-provided demograp
#          "Adjust the inputs in the sidebar to get a predicted insurance charge, a

# # 1. Load the dataset
# @st.cache_data
# def load_data():
#     df = pd.read_csv('medical_insurance.csv')
#     df = df.drop_duplicates()  # remove duplicate rows if any
#     return df

# df = load_data()

# # 2. Data preprocessing
# # Create BMI Category feature
# bins = [0, 18.5, 25, 30, float('inf')]
# labels = ['Underweight', 'Normal', 'Overweight', 'Obese']
# df['BMI_Category'] = pd.cut(df['bmi'], bins=bins, labels=labels, include_lowest=T

# # Encode binary categorical features
# # Sex: female->0, male->1; Smoker: no->0, yes->1
# df['sex'] = df['sex'].map({'female': 0, 'male': 1})
# df['smoker'] = df['smoker'].map({'no': 0, 'yes': 1})
```

```
# # One-hot encode region and BMI_Category
# df_encoded = pd.get_dummies(df, columns=['region', 'BMI_Category'])
# # Prepare feature matrix X and target vector y
# X = df_encoded.drop('charges', axis=1)
# y = df_encoded['charges']

# # 3. Train the regression model (XGBoost Regressor)
# model = XGBRegressor(n_estimators=100, random_state=0,
#                      use_label_encoder=False, eval_metric='rmse')
# model.fit(X, y)  # Train on the entire dataset

# # 4. Sidebar inputs for user to provide new data
# st.sidebar.header("Input Patient Data")
# # Age input
# age = st.sidebar.slider("Age", min_value=int(df['age'].min()), max_value=int(df['
# # Sex input
# sex_option = st.sidebar.selectbox("Sex", options=["Female", "Male"])
# # BMI input
# bmi = st.sidebar.slider("BMI", min_value=float(df['bmi'].min()), max_value=float(
# # Children input
# children = st.sidebar.number_input("Number of Children", min_value=0, max_value=i
# # Smoker input
# smoker_option = st.sidebar.selectbox("Smoker", options=["No", "Yes"])
# # Region input
# region_option = st.sidebar.selectbox("Region", options=[region.capitalize() for r
# # (Capitalize region names for display, but will convert back to lowercase for mo

# # 5. Transform user input to match training data encoding
# # Convert categorical inputs to encoded values
# sex_encoded = 1 if sex_option == "Male" else 0
# smoker_encoded = 1 if smoker_option == "Yes" else 0
# region_lower = region_option.lower()  # e.g., "Northwest" -> "northwest"

# # Determine BMI category for the input BMI (using same thresholds as above)
# if bmi < 18.5:
#     bmi_cat = 'Underweight'
# elif bmi < 25:
#     bmi_cat = 'Normal'
# elif bmi < 30:
#     bmi_cat = 'Overweight'
# else:
#     bmi_cat = 'Obese'

# # Initialize a dict with all required feature columns set to 0
# input_data = {col: 0 for col in X.columns}

# # Fill in the values for the input_data dict
# input_data['age'] = age
# input_data['bmi'] = bmi
# input_data['children'] = children
# input_data['sex'] = sex_encoded
# input_data['smoker'] = smoker_encoded
# # Set the appropriate region and BMI category dummy to 1
# region_col = f"region_{region_lower}"
# bmi_col = f"BMI_Category_{bmi_cat}"
```

```
# if region_col in input_data:
#     input_data[region_col] = 1
# if bmi_col in input_data:
#     input_data[bmi_col] = 1

# # Convert the input_data into a DataFrame
# input_df = pd.DataFrame([input_data])

# # 6. Predict using the trained model
# prediction = model.predict(input_df)[0]
# st.subheader("Predicted Insurance Charge:")
# st.write(f"**${prediction:,.2f}**")  # display prediction with 2 decimal places a

# # 7. Optional EDA visualizations
# st.markdown("___")  # horizontal rule for separation
# st.subheader("Explore the Data")
# st.write("Use the checkboxes below to view various **exploratory charts** about t

# # Checkbox 1: Distribution of Charges
# if st.checkbox("Show distribution of insurance charges"):
#     fig1, ax1 = plt.subplots(figsize=(6,4))
#     sns.histplot(df['charges'], kde=True, color='teal', ax=ax1)
#     ax1.set_title("Distribution of Insurance Charges")
#     ax1.set_xlabel("Charges")
#     ax1.set_ylabel("Frequency")
#     st.pyplot(fig1)

# # Checkbox 2: Charges by Smoking Status
# if st.checkbox("Compare charges: Smokers vs. Non-smokers"):
#     fig2, ax2 = plt.subplots(figsize=(6,4))
#     sns.boxplot(x='smoker', y='charges', data=load_data(), palette="Set2", ax=ax2
#     ax2.set_title("Insurance Charges by Smoking Status")
#     ax2.set_xlabel("Smoker (0 = No, 1 = Yes)")
#     ax2.set_ylabel("Charges")
#     st.pyplot(fig2)

# # Checkbox 3: BMI vs Charges scatter plot
# if st.checkbox("Show BMI vs. Charges scatterplot (colored by smoking status)"):
#     fig3, ax3 = plt.subplots(figsize=(6,4))
#     sns.scatterplot(x='bmi', y='charges', hue='smoker', data=load_data(), palette
#     ax3.set_title("BMI vs Insurance Charges")
#     ax3.set_xlabel("BMI")
#     ax3.set_ylabel("Charges")
#     ax3.legend(title="Smoker")
#     st.pyplot(fig3)
```