1. pH Level:

The pH level tells us whether water is acidic or alkaline. The World Health Organization (WHO) suggests that safe drinking water should have a pH between 6.5 and 8.5. Our tests show levels between 6.52 and 6.83, which is within the safe range.

2. Hardness:

Water hardness is due to minerals like calcium and magnesium. If water spends more time around rocks and soil containing these minerals, it becomes harder. Hard water can cause soap to not foam easily.

3. Solids (TDS - Total Dissolved Solids):

Water can dissolve many minerals and salts. If water tastes odd or looks colored, it might have high levels of these dissolved solids. For drinking, the desired limit is below 500 mg/l, but it's still acceptable up to 1000 mg/l.

4. Chloramines:

To make water safe to drink, we often add chlorine or chloramines. Chloramines form when we mix chlorine with ammonia. A chlorine level below 4 mg/L is considered safe for drinking.

5. Sulfate:

Sulfates are everywhere—in the air, soil, and our food. They're used a lot in the chemical industry. While seawater has about 2,700 mg/L of sulfate, freshwater usually has between 3 and 30 mg/L. Some places might have even higher levels.

6. Conductivity:

Pure water doesn't conduct electricity well. However, when there are more dissolved solids in the water, it conducts electricity better. The conductivity, or EC, tells us about this. The WHO recommends an EC value below 400 µS/cm.

7. Organic Carbon:

Organic carbon in water can come from natural sources like decaying plants or from human-made sources. The US EPA suggests that drinking water should have less than 2 mg/L of Total Organic Carbon (TOC), and source water used for treatment should have less than 4 mg/L.

8. Trihalomethanes (THMs):

THMs form in water that's been treated with chlorine. The amount of THMs can depend on several factors. However, as long as there are fewer than 80 ppm of THMs, the water is considered safe.

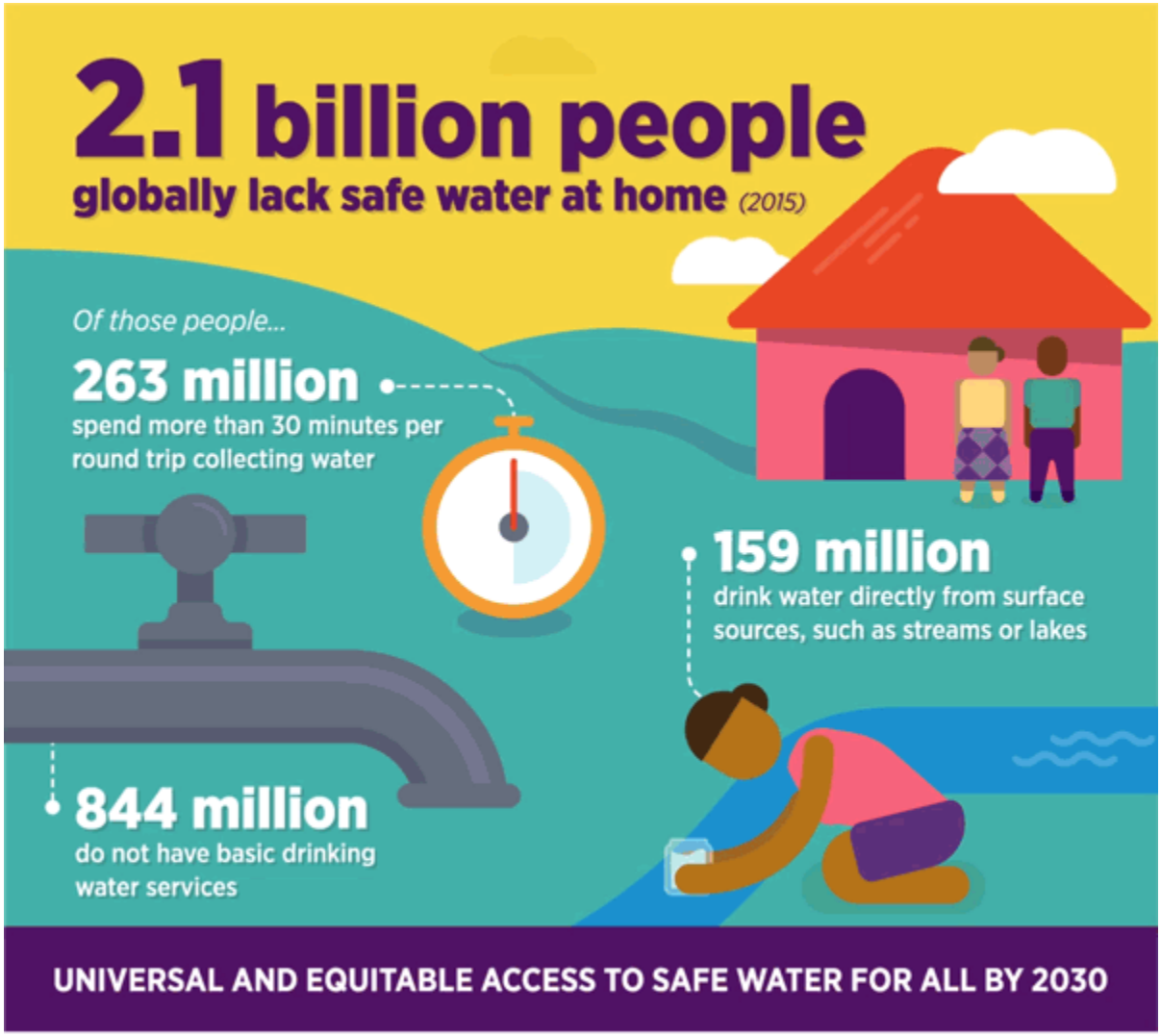
9. Turbidity:

Turbidity tells us about the clarity of water. If water has many suspended particles, it's more turbid. A test from the Wondo Genet Campus showed turbidity at 0.98 NTU, which is clearer than the WHO's safe limit of 5.00 NTU.

10. Potability:

This means where is drinkable or not. If it's labeled "1", it's drinkable. If it's "0", it's not.

In [1]:
```
from IPython.core.display import display, HTML

html_content = """
<div style="background-color:#B4DBE9;">
    <center><img src="https://raw.githubusercontent.com/jaykumar1607/Water-Qualit
</div>
"""

display(HTML(html_content))
```



World Health Organization unicef

Clean water is essential for health and well-being. Better water management can boost a country's growth and reduce poverty. Dirty water and bad sanitation lead to diseases like cholera and diarrhea. In many hospitals, especially in poorer countries, patients often get infections due to unclean conditions.

Inspired by this, I used a Water Quality dataset and machine learning to identify drinkable versus non-drinkable water.

In [2]:
```
# Data Analysis Libaries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
# Importing dataset
df = pd.read_csv('water_potability.csv')
```

In [4]:
```
# Checking out first 5 rows
df.head()
```

Out[4]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trih
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	

In [5]:
```
# Checking for data types and information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

In [6]:
```
df.shape
# 3276 rows and 10 columns with 1 iutout variable
```

Out[6]:
```
(3276, 10)
```

In [7]:
```
df.describe()
```

Out[7]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_ca
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.00
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.28
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.30
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.20
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.06
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.21
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.55
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.30

In [8]:
```
df.duplicated().sum()
# No duplicates at all
```

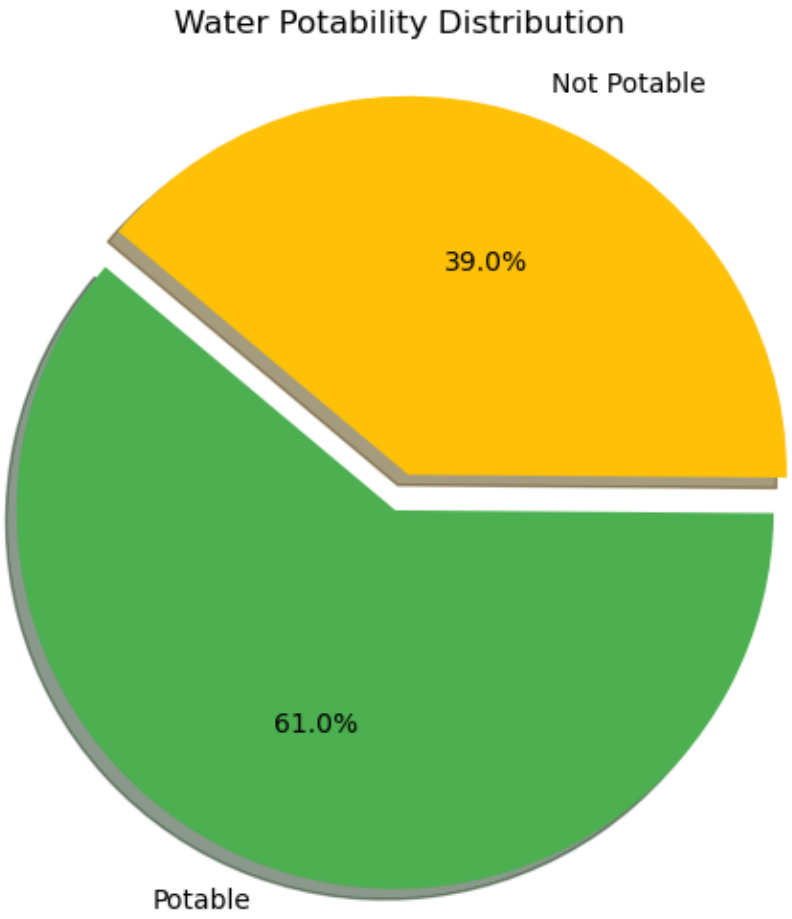
Out[8]:
```
0
```

In [9]:
```
df['Potability'].value_counts(normalize=True)
```

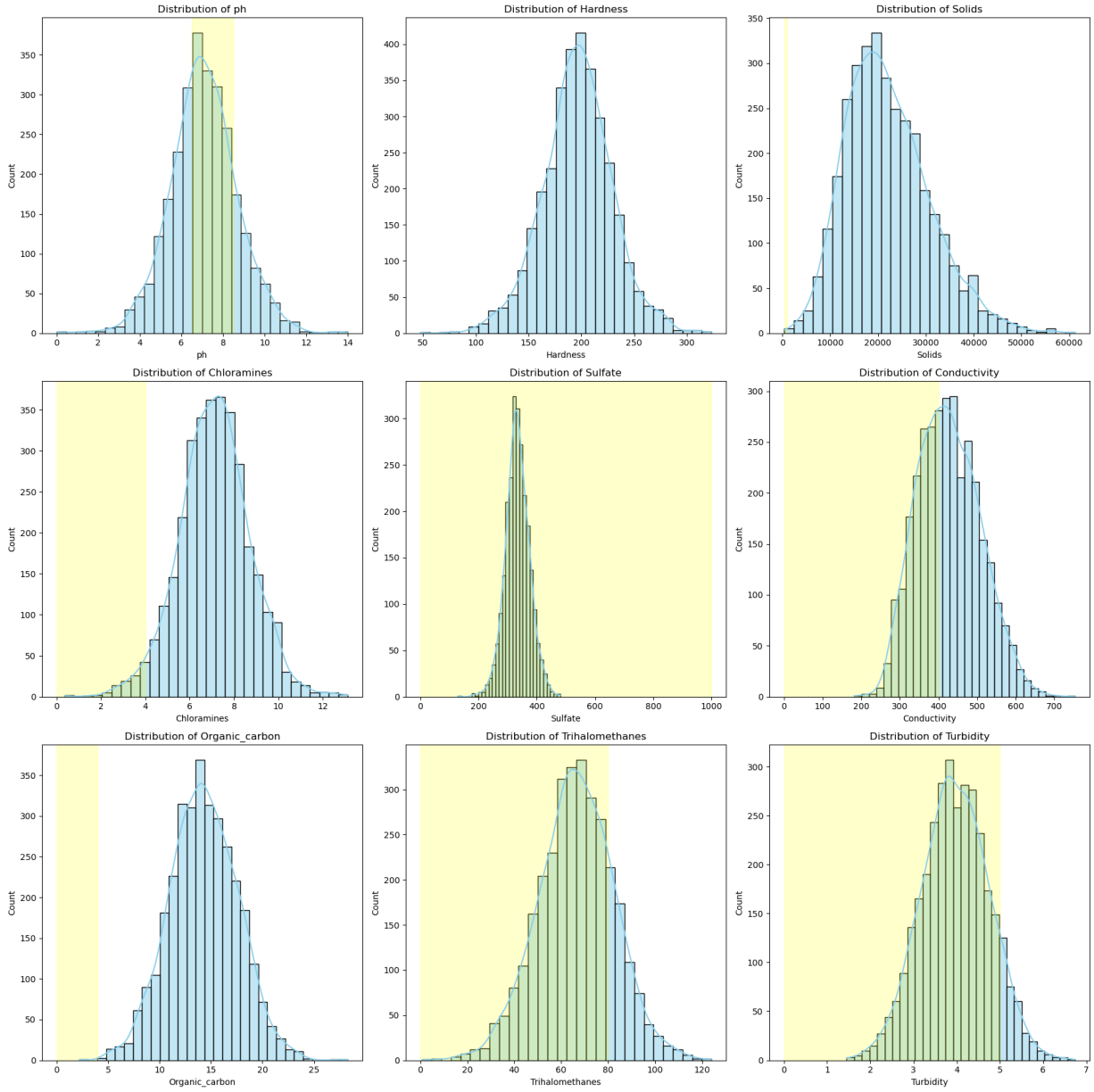
Out[9]:
```
0    0.60989
1    0.39011
Name: Potability, dtype: float64
```

About 61% of our target variable is "Potable" and 39% is "No Potable"

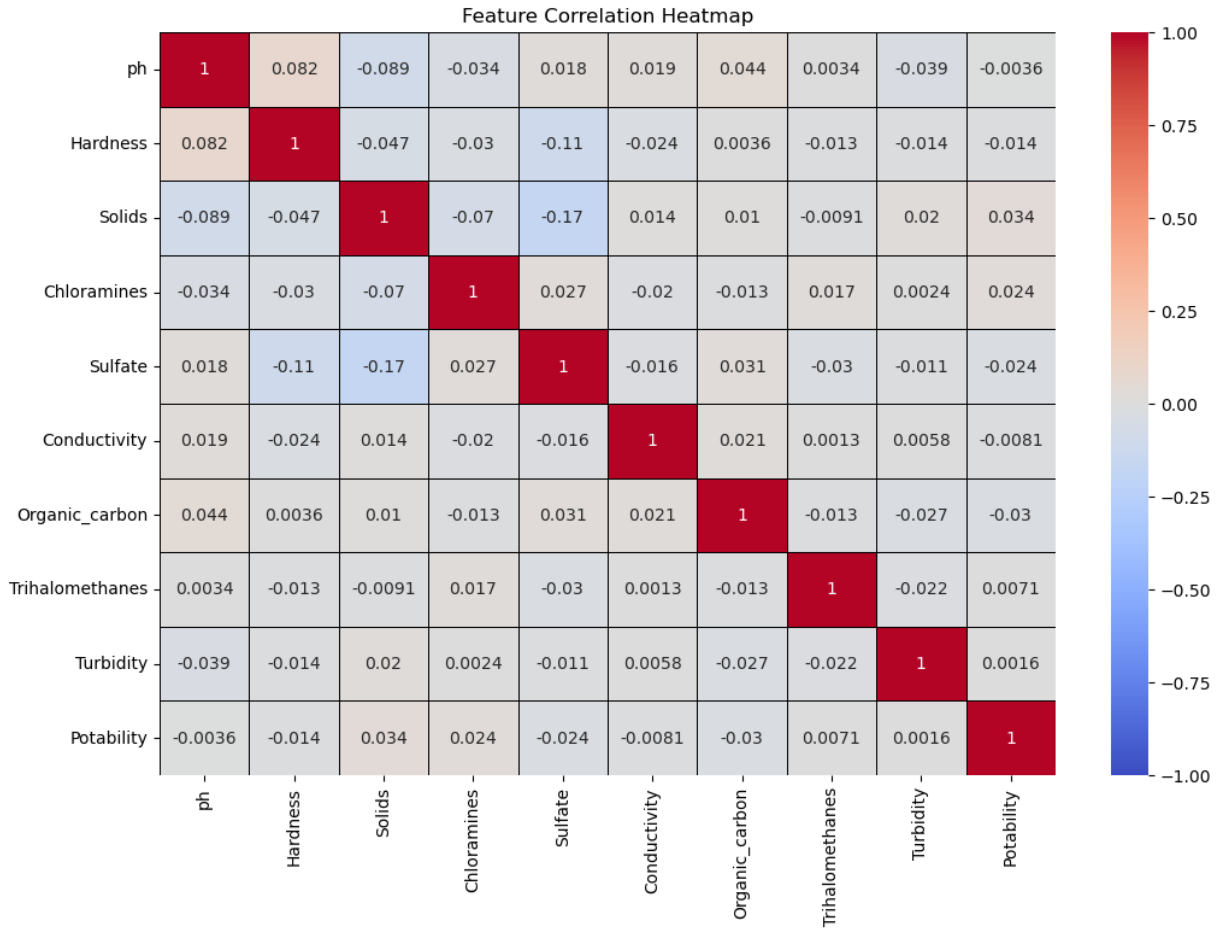
In [10]:
```
potability_counts = df['Potability'].value_counts(normalize=True)

# Create a pie chart
labels = ['Potable', 'Not Potable']
colors = ['#4CAF50', '#FFC107']
explode = (0.1, 0)  # explode 1st slice for emphasis

plt.figure(figsize=(8, 6))
plt.pie(potability_counts, explode=explode, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=140)
plt.title('Water Potability Distribution')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



In [11]:
```
limits = {
    "ph": (6.5, 8.5),
    "Solids": (500, 1000),
    "Chloramines": (0, 4),
    "Sulfate": (0, 1000),
    "Conductivity": (0, 400),
    "Organic_carbon": (0, 4),
    "Trihalomethanes": (0, 80),
    "Turbidity": (0, 5)
}
features = df.columns[:-1]
# Plotting the distributions with the defined limits
plt.figure(figsize=(18, 18))

for i, feature in enumerate(features, 1):
    plt.subplot(3, 3, i)
    sns.histplot(df[feature], bins=30, kde=True, color="skyblue")
    plt.title(f'Distribution of {feature}')
    if feature in limits:
        plt.axvspan(limits[feature][0], limits[feature][1], color='yellow', alpha
    plt.tight_layout()

plt.show()
```


Distribution of ph
Count
ph
Distribution of Hardness
Count
Hardness
Distribution of Solids
Count
Solids
Distribution of Chloramines
Count
Chloramines
Distribution of Sulfate
Count
Sulfate
Distribution of Conductivity
Count
Conductivity
Distribution of Organic_carbon
Count
Organic_carbon
Distribution of Trihalomethanes
Count
Trihalomethanes
Distribution of Turbidity
Count
Turbidity

In [12]:
```
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1, lir
plt.title("Feature Correlation Heatmap")
plt.show()
```



ph: The distribution appears somewhat normal, centered around the pH level of 7 (which is neutral). There are some outliers on both ends. Hardness: The distribution appears somewhat normal. Solids: The distribution is right-skewed, indicating that most samples have lower concentrations of dissolved solids. Chloramines: The distribution appears somewhat normal, centered around 7. Sulfate: The distribution appears somewhat normal, but there's a gap in the distribution around the average value. Conductivity: The distribution appears somewhat normal, with a few outliers on the higher end. Organic_carbon: The distribution appears somewhat normal. Trihalomethanes: The distribution appears somewhat normal. Turbidity: The distribution appears somewhat normal, centered around 4.

In [13]:
```
# Replacing missing values using the median
df['ph'].fillna(df['ph'].median(), inplace=True)
df['Sulfate'].fillna(df['Sulfate'].median(), inplace=True)
df['Trihalomethanes'].fillna(df['Trihalomethanes'].median(), inplace=True)

# Checking if there are any more missing values
remaining_missing = df.isnull().sum()

remaining_missing
```

Out[13]:
```
ph                0
Hardness          0
Solids            0
Chloramines       0
Sulfate           0
Conductivity      0
Organic_carbon    0
Trihalomethanes   0
Turbidity         0
Potability        0
dtype: int64
```

In [14]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Defining features and target variable
X = df.drop('Potability', axis=1)
y = df['Potability']

# Scaling the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, r

X_train.shape, X_test.shape
```

Out[14]:
```
((2620, 9), (656, 9))
```

In [15]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Initializing the models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42),
    "SVM": SVC(random_state=42)
}

# Training and evaluating the models
model_accuracies = {}

for name, model in models.items():
    # Training the model
    model.fit(X_train, y_train)

    # Predicting on the test set
    predictions = model.predict(X_test)

    # Calculating accuracy
    accuracy = accuracy_score(y_test, predictions)
    model_accuracies[name] = accuracy

model_accuracies
```

Out[15]:
```
{'Logistic Regression': 0.6280487804878049,
 'Random Forest': 0.6737804878048781,
 'Gradient Boosting': 0.6692073170731707,
 'SVM': 0.6920731707317073}
```

In [16]:
```
# Visualization for Model Evaluation: Model Accuracies
plt.figure(figsize=(12, 6))

# Bar plot of model accuracies
sns.barplot(x=list(model_accuracies.keys()), y=list(model_accuracies.values()), p
plt.title("Model Accuracies on Test Set")
plt.ylabel("Accuracy")
plt.xticks(rotation=45)
plt.ylim(0.5, 0.8)  # setting y-axis limits to better visualize differences

plt.tight_layout()
plt.show()
```