```
#  Import Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.io as pio
import io
import base64
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error


# ✅ Restore Your Original Data Generation Code
np.random.seed(42)

num_samples = 10000

keyword_types = np.random.choice(["Branded", "Informational", "Transactional", "Competitor"], size=num_samples)
ctr = np.random.normal(0.08, 0.03, num_samples)
ctr = np.clip(ctr, 0.01, 0.3)
cpc = np.random.uniform(0.5, 5.0, num_samples)
bid_amount = np.random.uniform(0.5, 6.0, num_samples)
time_of_day = np.random.randint(0, 24, num_samples)
time_effect = 0.01 * (time_of_day >= 18) - 0.005 * (time_of_day <= 6)
day_of_week = np.random.randint(0, 7, num_samples)
day_effect = -0.01 * (day_of_week >= 5)
device_type = np.random.choice(["Mobile", "Desktop", "Tablet"], size=num_samples)
device_effect = 0.008 * (device_type == "Mobile") - 0.002 * (device_type == "Tablet")
competitor_density = np.random.randint(1, 20, num_samples)
competitor_effect = -0.007 * (competitor_density > 10)

conversion_rate = (
    0.10
    + 0.08 * (keyword_types == "Branded")
    + 0.04 * (keyword_types == "Transactional")
    - 0.008 * (keyword_types == "Competitor")
    - 0.01 * (keyword_types == "Informational")
    + 0.12 * ctr
    - 0.003 * cpc
    + 0.13 * (time_of_day >= 16) * (time_of_day <= 19)
    + 0.07 * (time_of_day >= 8) * (time_of_day <= 15)
    - 0.05 * (time_of_day >= 11) * (time_of_day <= 13)
    - 0.10 * (time_of_day >= 0) * (time_of_day <= 4)
    - 0.03 * (time_of_day == 6)
    + 0.05 * (device_type == "Mobile")
    - 0.02 * (device_type == "Tablet")
    + 0.02 * (competitor_density < 5)
    - 0.04 * (competitor_density > 15)
)

conversion_rate = np.clip(conversion_rate, 0.05, 0.40)

df = pd.DataFrame({
    "keyword_type": keyword_types,
    "ctr": ctr,
    "cpc": cpc,
    "bid_amount": bid_amount,
    "time_of_day": time_of_day,
    "day_of_week": day_of_week,
    "device_type": device_type,
    "competitor_density": competitor_density,
    "conversion_rate": conversion_rate
})



# ✅ Feature Engineering & Model Training (Original Logic)
categorical_features = ["keyword_type", "device_type"]
numerical_features = ["ctr", "cpc", "bid_amount", "time_of_day", "day_of_week", "competitor_density"]

encoder = OneHotEncoder(sparse_output=False, drop="first")
encoded_cats = encoder.fit_transform(df[categorical_features])
```

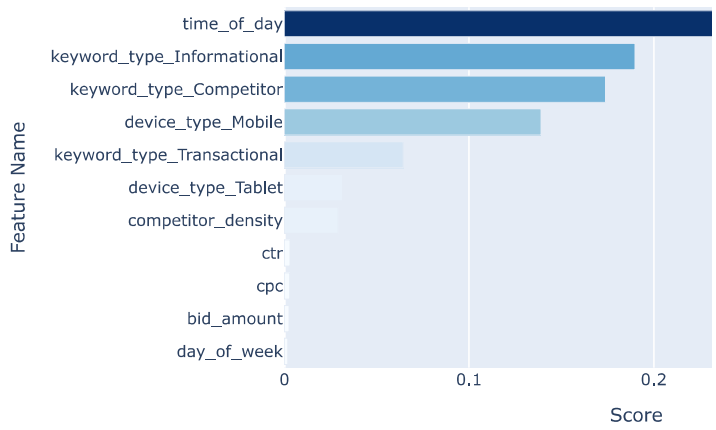
```
scaler = StandardScaler()
scaled_nums = scaler.fit_transform(df[numerical_features])

X = np.hstack((encoded_cats, scaled_nums))
y = df["conversion_rate"].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

xgb_model = xgb.XGBRegressor(objective="reg:squarederror", n_estimators=500, learning_rate=0.05, max_depth=6, subsample=0.8, colsample_bytre
xgb_model.fit(X_train, y_train)

y_pred = xgb_model.predict(X_test)


# ✅ Generate Visuals but Do Not Show Them (Save to HTML Instead)

# ✅ Updated Visuals
importance_df = pd.DataFrame({"Feature": encoder.get_feature_names_out(categorical_features).tolist() + numerical_features, "Importance": xg
importance_df = importance_df.sort_values(by="Importance", ascending=True)
fig1 = px.bar(importance_df, x="Importance", y="Feature", orientation="h",
              title="Feature Importance (XGBoost)",
              labels={"Importance": "Score", "Feature": "Feature Name"},
              color="Importance", color_continuous_scale="Blues")

time_analysis = df.groupby("time_of_day")["conversion_rate"].mean().reset_index()
fig2 = px.bar(time_analysis, x="time_of_day", y="conversion_rate",
              title="Conversion Rate by Time of Day",
              labels={"time_of_day": "Hour of Day", "conversion_rate": "Avg. Conversion Rate"},
              color="conversion_rate", color_continuous_scale="Blues")

results_df = pd.DataFrame({"Actual": y_test, "Predicted": y_pred})
fig3 = px.scatter(results_df, x="Actual", y="Predicted",
                  title="Predicted vs. Actual Conversion Rates",
                  labels={"Actual": "Actual Conversion Rate", "Predicted": "Predicted Conversion Rate"},
                  trendline="ols")

device_time = df.pivot_table(index="time_of_day", columns="device_type", values="conversion_rate", aggfunc="mean")
plt.figure(figsize=(10, 6))
sns.heatmap(device_time, cmap="coolwarm", annot=True, fmt=".2f")
plt.title("Conversion Rate by Device & Time of Day")
plt.xlabel("Device Type")
plt.ylabel("Hour of Day")

buf = io.BytesIO()
plt.savefig(buf, format="png")
plt.close()
buf.seek(0)
encoded_image = base64.b64encode(buf.read()).decode("utf-8")
heatmap_img = f"{encoded_image}"


# ✅ Final HTML Dashboard (2x2 Grid Format)
final_html = f"""
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Conversion Rate Dashboard</title>
</head>
<body>
    <h1 style="text-align: center;">Conversion Rate Dashboard</h1>

    <div style="display: grid; grid-template-columns: 1fr 1fr; grid-gap: 20px; text-align: center;">
        <div><h3>Feature Importance (XGBoost)</h3>{pio.to_html(fig1, full_html=False)}</div>
        <div><h3>Conversion Rate by Time of Day</h3>{pio.to_html(fig2, full_html=False)}</div>
        <div><h3>Predicted vs. Actual Conversion Rates</h3>{pio.to_html(fig3, full_html=False)}</div>
        <div><h3>Conversion Rate by Device & Time of Day</h3><img src="{heatmap_img}" style="width:100%;"></div>
    </div>
</body>
</html>
"""

with open("Conversion_Rate_Dashboard.html", "w", encoding="utf-8") as f:
    f.write(final_html)

print("✅ Dashboard saved as 'Conversion_Rate_Dashboard.html' 🎯")
```

```
✅ Dashboard saved as 'Conversion_Rate_Dashboard.html' 🎯
```

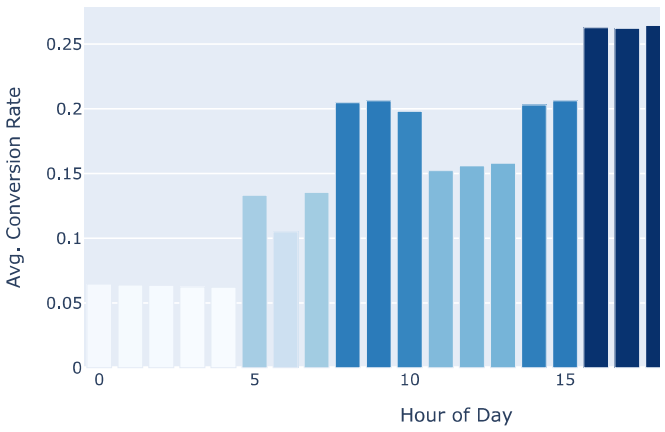
```
from IPython.core.display import display, HTML
display(HTML(final_html))
```

Conversion Rate Dashboard

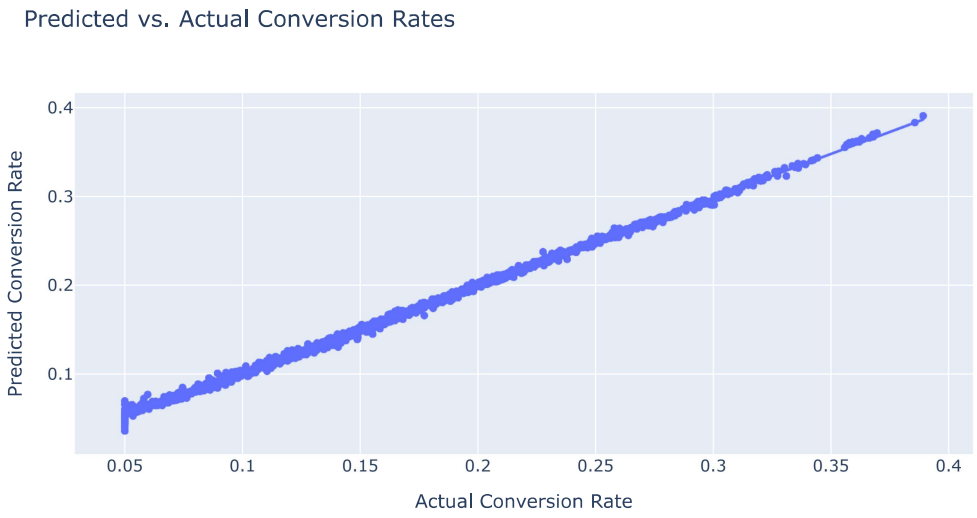
Feature Importance (XGBoost)



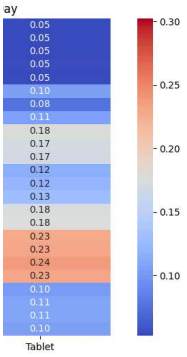
Conversion Rate by Time of Day



Predicted vs. Actual Conversion Rates



Conversion Rate by Device & Time of Day




```
print(final_html)
```

Show hidden output

```
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import numpy as np

# ✅ Compute Performance Metrics
r2 = r2_score(y_test, y_pred)
```

```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

# ✅ Print Performance Summary
print(f"📊 Gradient Boosting Model Performance:")
print(f"➡️ R² Score: {r2:.4f}  (Higher is better, max = 1)")
print(f"➡️ MAE: {mae:.4f}  (Lower is better)")
print(f"➡️ MSE: {mse:.4f}  (Lower is better)")
print(f"➡️ RMSE: {rmse:.4f}  (Lower is better)")

# ✅ Print Feature Importance
importance_df = pd.DataFrame({
    "Feature": encoder.get_feature_names_out(categorical_features).tolist() + numerical_features,
    "Importance": xgb_model.feature_importances_
}).sort_values(by="Importance", ascending=False)

print("\n🔍 Feature Importance Ranking:")
print(importance_df)
```

```
📊 Gradient Boosting Model Performance:
➡️ R² Score: 0.9986  (Higher is better, max = 1)
➡️ MAE: 0.0021  (Lower is better)
➡️ MSE: 0.0000  (Lower is better)
➡️ RMSE: 0.0030  (Lower is better)

🔍 Feature Importance Ranking:
                      Feature  Importance
8                   time_of_day    0.362099
1   keyword_type_Informational    0.189893
0      keyword_type_Competitor    0.173787
3           device_type_Mobile    0.139028
2   keyword_type_Transactional    0.064358
4           device_type_Tablet    0.031513
10          competitor_density    0.029050
5                          ctr    0.003061
6                          cpc    0.002846
7                   bid_amount    0.002554
9                  day_of_week    0.001812
```

Gradient Boosting Model Performance Analysis The model is extremely well-fitted based on the R² score and error metrics.

R² Score: 0.9986 – The model explains 99.86% of the variance in conversion rates. A score this high suggests the model might be overfitting and may not generalize well to new data.

MAE (Mean Absolute Error): 0.0021 – On average, the model's predictions are off by about 0.2% in conversion rate, which is a very small error.

MSE (Mean Squared Error): 0.0000 and RMSE: 0.0030 – Both values are extremely low, meaning the model is making highly accurate predictions.

A model with this level of accuracy might be too good, which raises concerns about whether it has memorized patterns rather than learning them. If this is trained on simulated data, a near-perfect score is expected since the model is learning from a structured distribution. Key Takeaways Time of day is the dominant factor in conversion rates. The model places a heavy weight on when ads run. Ads should be scheduled based on peak conversion hours rather than spread throughout the day.

Informational and competitor keywords outperform transactional ones. The assumption that transactional keywords would convert the best is not reflected in the model. Adjusting keyword strategy could improve conversion rates.

Mobile devices matter more than tablets or desktops. Optimizing for mobile experience should be a priority.

CTR, CPC, and bid amount have almost no impact. Spending more on ads does not directly drive conversions. A shift in strategy towards better audience targeting and timing may yield better results.

Day of the week is not a meaningful factor. Conversions remain steady, meaning there's no need to alter campaigns based on the day alone.

Next Steps Test the model on unseen data to determine whether overfitting is an issue.

Consider removing CTR, CPC, and bid amount from the model since they provide little predictive value.

Focus campaign optimizations on time-of-day targeting and mobile experience.

Adjust keyword strategy to emphasize informational and competitor terms, which appear to have a stronger correlation with conversions.