assignment-1-ds

April 17, 2025

[14]:
```
import pandas as pd
import numpy as np
```

[4]:
```
#identify null value
df=pd.read_csv("C:\\Users\\DELL\\Desktop\\razifa\\tips.csv")
print(df.isnull().sum())
```

```
total_bill    0
tip           0
sex           3
smoker        1
day           6
time          4
size          0
dtype: int64
```

[7]:
```
#Handling Null Values
df_clean=df.dropna()
df_clean=df.dropna(axis=1)
print(df_clean)
```

```
     total_bill   tip  size
0         16.99  1.01     2
1         10.34  1.66     3
2         21.01  3.50     3
3         23.68  3.31     2
4         24.59  3.61     4
..          …    …    …
239       29.03  5.92     3
240       27.18  2.00     2
241       22.67  2.00     2
242       17.82  1.75     2
243       18.78  3.00     2

[244 rows x 3 columns]
```

[9]:
```
# fill Null value(Imputation)
df.fillna(0,inplace=True)
```

```
df
```

[9]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
..	…	…	…	…	…	…	…
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

[244 rows x 7 columns]

[10]:
```
df.fillna(df.mode().iloc[0], inplace=True)
df
```

[10]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
..	…	…	…	…	…	…	…
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

[244 rows x 7 columns]

[16]:
```
#identifying outliers
# Using Z-score method
from scipy import stats
z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
outliers = (z_scores > 3).sum()
print("Outliers in each column:\n", outliers)
```

```
Outliers in each column:
 total_bill    4
tip            3
size           4
dtype: int64
```

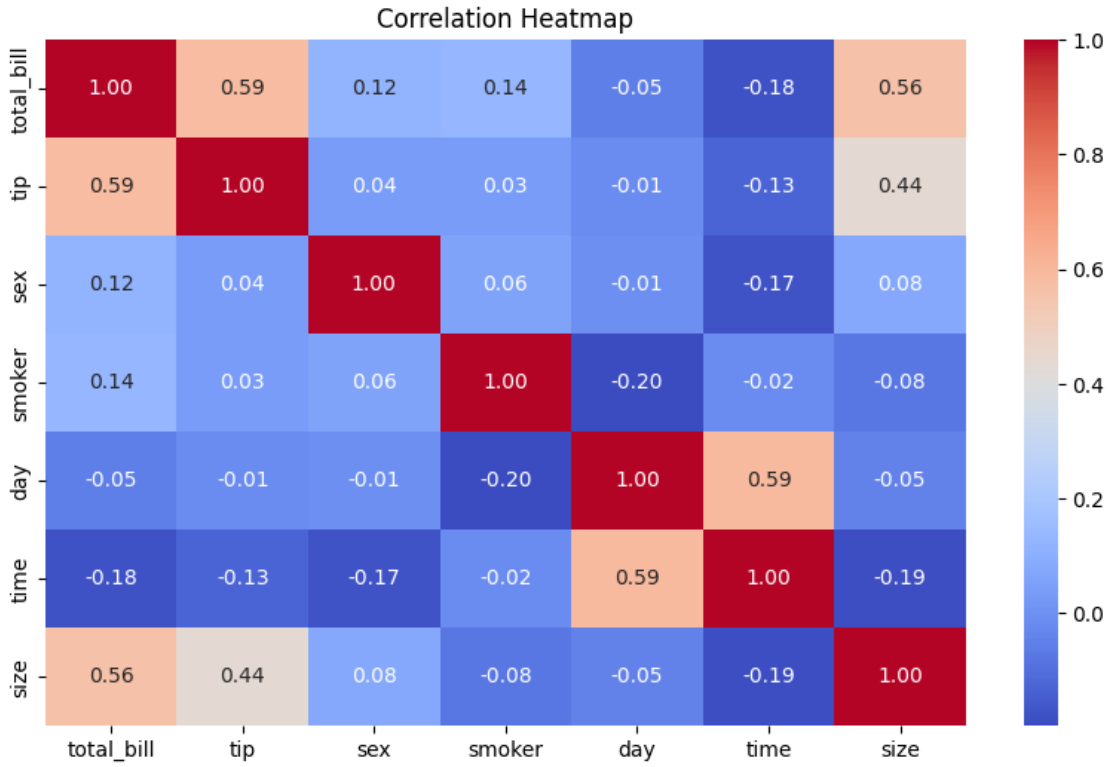
[18]:
```
# Removing outliers where Z-score > 3
df_no_outliers = df[(z_scores < 3).all(axis=1)]
df_no_outliers
```

[18]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
..	…	…	…	…	…	…	…
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

[236 rows x 7 columns]

[25]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Copy the dataset before encoding
df_encoded = df_no_outliers.copy()

# Encode categorical columns
for col in df_encoded.select_dtypes(include=['object']).columns:
    df_encoded[col] = LabelEncoder().fit_transform(df_encoded[col].astype(str))

# Compute correlation matrix
correlation_matrix = df_encoded.corr()

# Plot Correlation Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



[27]:
```
null_hypothesis = "Tip amount does not depend on total_bill."
alternate_hypothesis = "Tip amount depends on total_bill."
```

[44]:
```
null_hypothesis = "Tip amount does not depend on total_bill."
alternate_hypothesis = "Tip amount depends on total_bill."
print("Null Hypothesis:", null_hypothesis)
print("Alternate Hypothesis:", alternate_hypothesis)
```

```
Null Hypothesis: Tip amount does not depend on total_bill.
Alternate Hypothesis: Tip amount depends on total_bill.
```

[45]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X = df_no_outliers[['total_bill', 'size']]
y = df_no_outliers['tip']

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

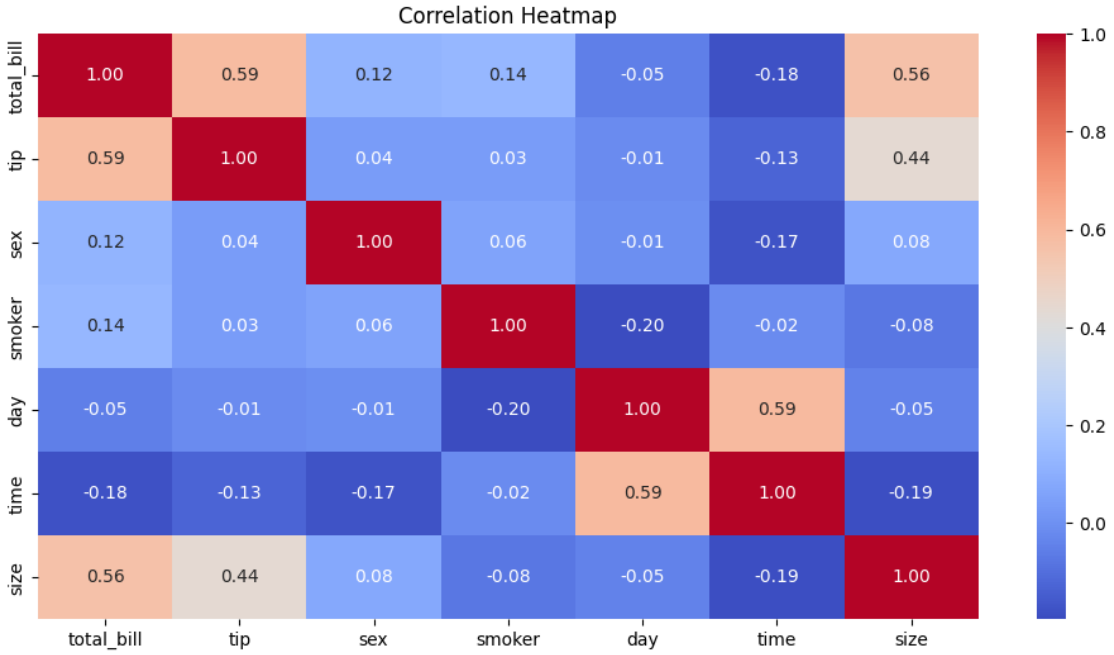
[29]:
```
# Training Linear Regression Model
model = LinearRegression()
```

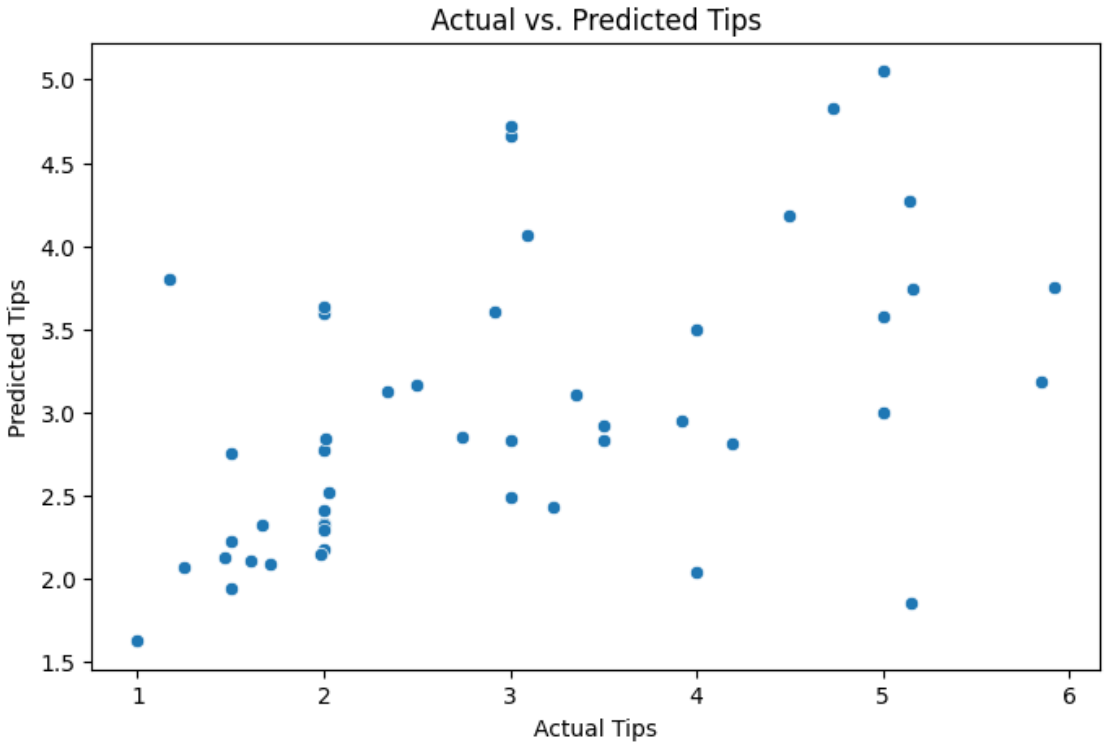
```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

[30]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("MAE:", mae)
print("MSE:", mse)
print("R² Score:", r2)
```

```
MAE: 0.9256223461741296
MSE: 1.4029515729711761
R² Score: 0.245709172450712
```

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

# Heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()

# Scatterplot of Actual vs. Predicted Tips
plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_test, y=y_pred)
plt.xlabel("Actual Tips")
plt.ylabel("Predicted Tips")
plt.title("Actual vs. Predicted Tips")
plt.show()
```


Correlation Heatmap
total_bill
tip
sex
smoker
day
time
size
1.00 0.59 0.12 0.14 -0.05 -0.18 0.56
0.59 1.00 0.04 0.03 -0.01 -0.13 0.44
0.12 0.04 1.00 0.06 -0.01 -0.17 0.08
0.14 0.03 0.06 1.00 -0.20 -0.02 -0.08
-0.05 -0.01 -0.01 -0.20 1.00 0.59 -0.05
-0.18 -0.13 -0.17 -0.02 0.59 1.00 -0.19
0.56 0.44 0.08 -0.08 -0.05 -0.19 1.00
1.0
0.8
0.6
0.4
0.2
0.0


Actual vs. Predicted Tips
Predicted Tips
Actual Tips
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1
2
3
4
5
6

[32]:
```
t_stat, p_value = stats.ttest_ind(df_no_outliers['total_bill'],␣
 ↪df_no_outliers['tip'])
print(f"T-Statistic: {t_stat}, P-Value: {p_value}")
```

```
T-Statistic: 30.837720796336914, P-Value: 5.474145811342345e-115
```

[37]:
```
#using random forest model
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder

# Encode categorical columns
df_encoded = df_no_outliers.copy()
for col in df_encoded.select_dtypes(include=['object']).columns:
    df_encoded[col] = LabelEncoder().fit_transform(df_encoded[col].astype(str))

# Features & Target
X = df_encoded[['total_bill', 'size']]
y = df_encoded['tip']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)



# Evaluation
print("Random Forest R²:", r2_score(y_test, y_pred_rf))
```

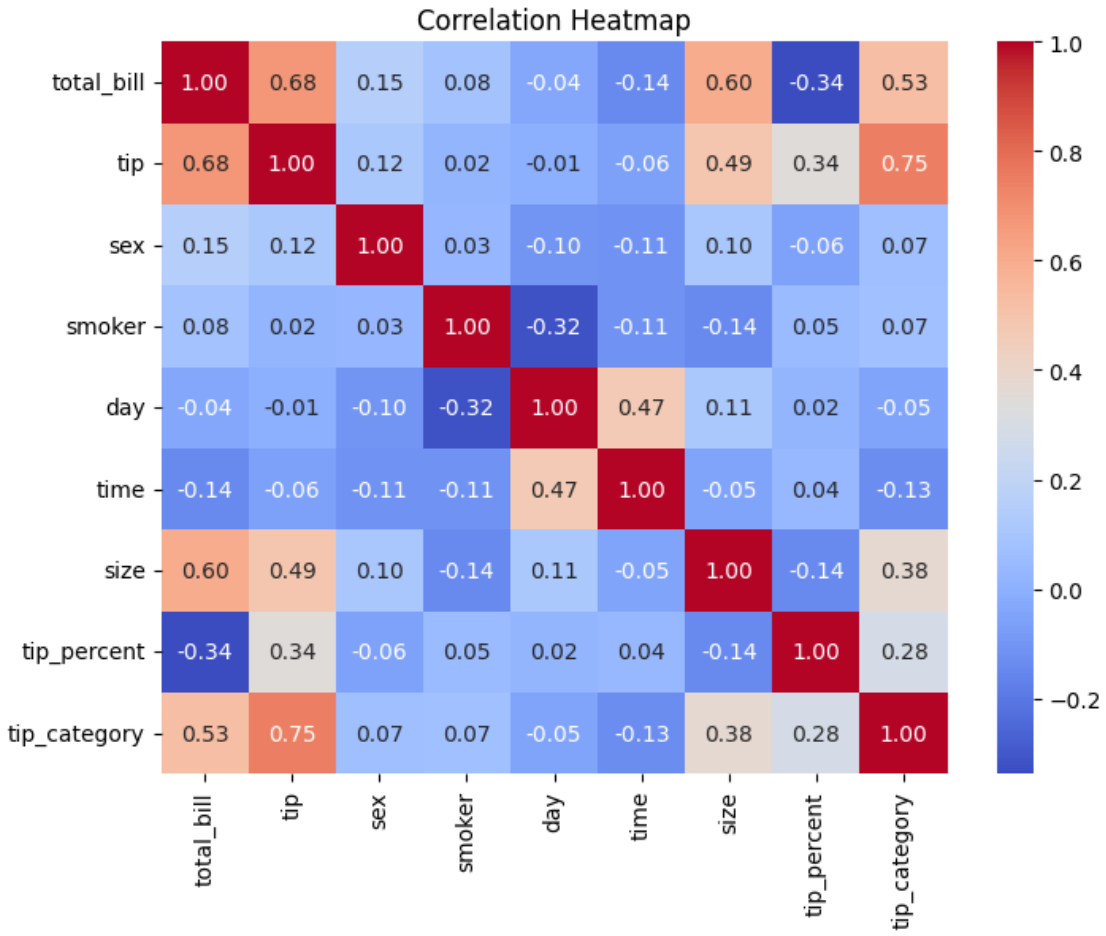
```
Random Forest R²: 0.045028230117261425
```

[40]:
```
#using random forest and logistic regression model
# increasing R2 score
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

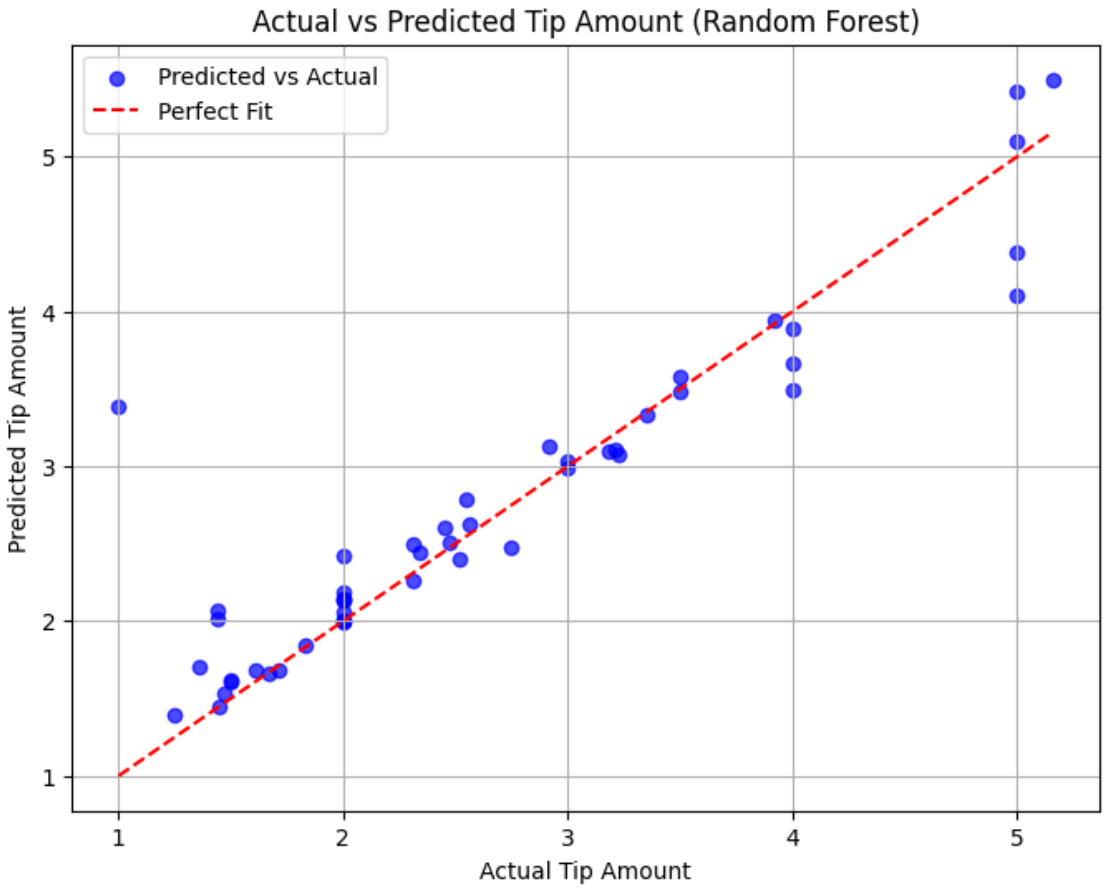
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score,␣
 ↪accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv('C:\\Users\\DELL\\Desktop\\razifa\\tips.csv')

# Encode categorical columns
df_encoded = df.copy()
for col in df_encoded.select_dtypes(include=['object']).columns:
    df_encoded[col] = LabelEncoder().fit_transform(df_encoded[col].astype(str))

# Handle missing values (Fill numeric columns with median)
df_encoded.fillna(df_encoded.median(numeric_only=True), inplace=True)

# **Feature Engineering**: Create a new column "tip_percent"
df_encoded['tip_percent'] = df_encoded['tip'] / df_encoded['total_bill']

# **Optimized Feature Selection**
features = ['total_bill', 'size', 'sex', 'smoker', 'day', 'time', 'tip_percent']
X = df_encoded[features]
y = df_encoded['tip']  # Target variable

# Train-Test Split (80-20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

### **1 Random Forest Regression (Optimized)**
rf = RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42)  #␣
 ↪Increased trees & depth
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# **Evaluation**
print("Optimized Random Forest Regression Metrics:")
print("MAE:", mean_absolute_error(y_test, y_pred_rf))
print("MSE:", mean_squared_error(y_test, y_pred_rf))
print("R² Score:", r2_score(y_test, y_pred_rf))  # Should be improved!

### **2 Logistic Regression (Classifying Tip as High/Low)**
df_encoded['tip_category'] = (df_encoded['tip'] > df_encoded['tip'].median()).
 ↪astype(int)  # Convert tip to 0/1
y_class = df_encoded['tip_category']  # Binary target for classification

# Train-Test Split for classification
X_train, X_test, y_train_class, y_test_class = train_test_split(X, y_class,␣
 ↪test_size=0.2, random_state=42)
```

```
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train_class)
y_pred_log_reg = log_reg.predict(X_test)

# **Evaluation**
print("\nLogistic Regression Classification Metrics:")
print("Accuracy:", accuracy_score(y_test_class, y_pred_log_reg))

# **Plot Heatmap**
plt.figure(figsize=(8, 6))
sns.heatmap(df_encoded.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```

```
Optimized Random Forest Regression Metrics:
MAE: 0.2215111037544794
MSE: 0.18486191638581342
R² Score: 0.8521070766262504

Logistic Regression Classification Metrics:
Accuracy: 0.7551020408163265
```



[41]:
```
# Scatter plot: Actual vs. Predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_rf, alpha=0.7, color='blue', label='Predicted vs␣
 ↪Actual')

# Plot the perfect prediction line (y = x)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',␣
 ↪linestyle='--', label='Perfect Fit')

plt.xlabel("Actual Tip Amount")
plt.ylabel("Predicted Tip Amount")
plt.title("Actual vs Predicted Tip Amount (Random Forest)")
plt.legend()
plt.grid(True)
plt.show()
```



[42]:
```
# Create a new DataFrame with actual and predicted values
df_results = X_test.copy()
df_results['Actual_Tip'] = y_test
df_results['Predicted_Tip'] = y_pred_rf  # Add the predicted column

# Save to CSV
df_results.to_csv('C:\\Users\\DELL\\Desktop\\razifa\\tips.csv', index=False)
print("Dataset with predicted values saved successfully!")
```

```
Dataset with predicted values saved successfully!
```

[43]:
```
df_results
```

[43]:

	total_bill	size	sex	smoker	day	time	tip_percent	Actual_Tip	\
24	19.82	2	1	0	1	0	0.160444	3.18	
6	8.77	2	1	0	2	0	0.228050	2.00	
153	24.55	4	1	0	2	0	0.081466	2.00	
211	25.89	4	1	1	1	0	0.199305	5.16	
198	13.00	2	0	1	3	1	0.153846	2.00	

176	17.89	2	1	1	2	0	0.111794	2.00
192	28.44	2	1	1	3	1	0.090014	2.56
124	12.48	2	0	0	3	1	0.201923	2.52
9	14.78	2	1	0	2	0	0.218539	3.23
101	15.38	2	0	1	0	0	0.195059	3.00
45	18.29	2	1	0	2	0	0.164024	3.00
233	10.77	2	1	0	1	0	0.136490	1.47
117	10.65	2	0	0	3	1	0.140845	1.50
177	14.48	2	1	1	2	0	0.138122	2.00
82	10.07	1	0	0	3	1	0.181728	1.83
146	18.64	3	0	0	3	1	0.072961	1.36
200	18.71	3	1	1	3	1	0.213789	4.00
15	21.58	2	1	0	2	0	0.181650	3.92
66	16.45	2	0	0	1	0	0.150152	2.47
142	41.19	5	1	0	3	1	0.121389	5.00
33	20.69	4	0	0	1	0	0.118415	2.45
19	20.65	3	1	0	1	2	0.162228	3.35
109	14.31	2	0	1	1	0	0.279525	4.00
30	9.55	2	1	0	1	0	0.151832	1.45
186	20.90	3	0	1	2	0	0.167464	3.50
120	11.69	2	1	0	3	1	0.197605	2.31
10	10.27	2	1	0	2	0	0.166504	1.71
73	25.28	2	0	1	1	0	0.197785	5.00
159	16.49	4	1	0	2	0	0.121286	2.00
156	48.17	6	1	0	2	0	0.103799	5.00
112	38.07	3	1	0	2	0	0.105070	4.00
218	7.74	2	1	1	1	0	0.186047	1.44
25	17.81	4	1	0	4	0	0.131387	2.34
60	20.29	2	1	1	1	0	0.158206	3.21
18	16.97	3	0	0	4	0	0.206246	3.50
119	24.08	4	0	0	3	1	0.121262	2.92
97	12.03	2	1	1	0	0	0.124688	1.50
197	43.11	4	0	1	3	1	0.115982	5.00
139	13.16	2	0	0	3	1	0.208967	2.75
241	22.67	2	1	1	1	0	0.088222	2.00
75	10.51	2	1	0	1	0	0.118934	1.25
127	14.52	2	0	0	3	1	0.137741	2.00
113	23.95	2	1	0	2	0	0.106472	2.55
16	10.33	3	0	0	2	0	0.161665	1.67
196	10.34	2	1	1	3	1	0.193424	2.00
67	3.07	1	0	1	1	0	0.325733	1.00
168	10.59	2	0	1	1	0	0.152030	1.61
38	18.69	3	1	0	1	0	0.123596	2.31
195	7.56	2	1	0	3	1	0.190476	1.44

	Predicted_Tip
24	3.098095

6	2.424300
153	2.190997
211	5.499642
198	2.006314
176	2.139598
192	2.629431
124	2.400346
9	3.079307
101	2.992533
45	3.036022
233	1.529652
117	1.614665
177	2.001518
82	1.840122
146	1.699700
200	3.889810
15	3.939351
66	2.506150
142	4.378300
33	2.609123
19	3.334489
109	3.498350
30	1.441844
186	3.481593
120	2.256121
10	1.687243
73	5.421242
159	2.147413
156	5.098400
112	3.661050
218	2.011700
25	2.439327
60	3.107896
18	3.574000
119	3.128078
97	1.602672
197	4.100800
139	2.472279
241	2.128753
75	1.393017
127	1.994976
113	2.787267
16	1.665864
196	2.057255
67	3.385850
168	1.685000
38	2.497781

```
195       2.070600
```

[ ]: