WAITER TIPS PREDICTION

NAME	ROLL NUMBER
A.Suraj Reddy	AM.EN.U4CSE20012
A.S.K Viswas	AM.EN.U4CSE20013
N. Sai Sandeep	AM.EN.U4CSE20049
P. Laxmi Praneeth	AM.EN.U4CSE20052
C.G.S Pranav Advaith	AM.EN.U4CSE20056

MACHINE LEARNING PROJECT PHASE 2

1. Problem Definition

Giving tips to waiters depends on many factors like number of customers, bill they made, type of restaurant. So, in this project we develop a model from which we can analyse and predict the tips for the waiter.

2. Datasets

https://www.kaggle.com/datasets/aminizahra/tips-dataset

https://www.kaggle.com/datasets/rupakroy/waiter-tips-dataset-for-prediction

https://raw.githubusercontent.com/amankharwal/Website-data/master/tips.csv

The dataset contains attributes total_bill, tip, sex, smoker, day,time, size,price_per_person,payer name, cc number. (244,11) is shape of dataset1.

3.Prepare Data

- **Preprocessing**

  •We want to predict the tip from other columns therefore we have to scale the numerical columns and encode categorical columns. For binary ones we have to either use label encoding or one hot encode them, then drop duplicate ones.

In [58]:
```
from sklearn.preprocessing import LabelEncoder
le= LabelEncoder()
```

In [59]:
```
df["sex"]=pd.DataFrame(le.fit_transform(df["sex"]))
df["time"]=pd.DataFrame(le.fit_transform(df["time"]))
df["smoker"]=pd.DataFrame(le.fit_transform(df["smoker"]))
df
```

Out[59]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	0	0	Sun	0	2
1	10.34	1.66	1	0	Sun	0	3
2	21.01	3.50	1	0	Sun	0	3
3	23.68	3.31	1	0	Sun	0	2
4	24.59	3.61	0	0	Sun	0	4
...	...	...	...	...	...	...	...
201	12.16	2.20	1	1	Fri	1	2
202	13.42	3.48	0	1	Fri	1	2
203	8.58	1.92	1	1	Fri	1	1
204	15.98	3.00	0	0	Fri	1	3
205	13.42	1.58	1	1	Fri	1	2

206 rows × 7 columns

Technique For Multi Categorical Variables

The technique is that we will limit one-hot encoding to the 10 most frequent labels of the variable. Here we do One-hot encoding for day

First we make dummies for the day column i.e using get_dummies method from pandas.

In [60]:
```
dummie = pd.get_dummies(df["day"],prefix="day")
dummie
```

Out[60]:

	day_Fri	day_Sat	day_Sun	day_Thur
0	0	0	1	0
1	0	0	1	0
2	0	0	1	0
3	0	0	1	0
4	0	0	1	0
...	...	...	...	...
201	1	0	0	0
202	1	0	0	0
203	1	0	0	0
204	1	0	0	0
205	1	0	0	0

206 rows × 4 columns

Now lets append this to our DataFrame

In [61]:
```
df = pd.concat([df,dummie],axis=1)
df
```

Out[61]:

	total_bill	tip	sex	smoker	day	time	size	day_Fri	day_Sat	day_Sun	day_Thur
0	16.99	1.01	0	0	Sun	0	2	0	0	1	0
1	10.34	1.66	1	0	Sun	0	3	0	0	1	0
2	21.01	3.50	1	0	Sun	0	3	0	0	1	0
3	23.68	3.31	1	0	Sun	0	2	0	0	1	0
4	24.59	3.61	0	0	Sun	0	4	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...
201	12.16	2.20	1	1	Fri	1	2	1	0	0	0
202	13.42	3.48	0	1	Fri	1	2	1	0	0	0
203	8.58	1.92	1	1	Fri	1	1	1	0	0	0
204	15.98	3.00	0	0	Fri	1	3	1	0	0	0
205	13.42	1.58	1	1	Fri	1	2	1	0	0	0

206 rows × 11 columns

Dropping the Day column now

In [62]:
```
df = df.drop(["day"],axis=1)
```

In [63]:
```
df.head()
```

Out[63]:

	total_bill	tip	sex	smoker	time	size	day_Fri	day_Sat	day_Sun	day_Thur
0	16.99	1.01	0	0	0	2	0	0	1	0
1	10.34	1.66	1	0	0	3	0	0	1	0
2	21.01	3.50	1	0	0	3	0	0	1	0
3	23.68	3.31	1	0	0	2	0	0	1	0
4	24.59	3.61	0	0	0	4	0	0	1	0

Normalizing the numerical columns

Here the numerical columns which are needed to normalize are **total_bill,tip and size**

We use MinMaxScaler to normalize the numerical columns.

In [64]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

In [65]:
```
columns_to_normalize = ["total_bill","tip","size"]
scaled_columns = pd.DataFrame(scaler.fit_transform(df[columns_to_normalize]),columns=columns_to_normalize)
scaled_columns
```

Out[65]:

	total_bill	tip	size
0	0.291579	0.001111	0.2
1	0.152283	0.073333	0.4
2	0.375786	0.277778	0.4
3	0.431713	0.256667	0.2
4	0.450775	0.290000	0.6
...	...	...	...
201	0.190406	0.133333	0.2
202	0.216799	0.275556	0.2
203	0.115417	0.102222	0.0
204	0.270423	0.222222	0.4
205	0.216799	0.064444	0.2

206 rows × 3 columns

In [66]:
```
scaled_columns.describe()
```

Out[66]:

	total_bill	tip	size
count	206.000000	206.000000	206.000000
mean	0.355873	0.231645	0.315534
std	0.192528	0.158180	0.191659
min	0.000000	0.000000	0.000000
25%	0.216328	0.111111	0.200000
50%	0.311060	0.222222	0.200000
75%	0.450566	0.306389	0.400000
max	1.000000	1.000000	1.000000

Now lets concat this and drop the previous **total_bill,tip and size**

In [67]:
```
df = df.drop(["total_bill","tip","size"],axis=1)
df = pd.concat([df,scaled_columns],axis=1)
```

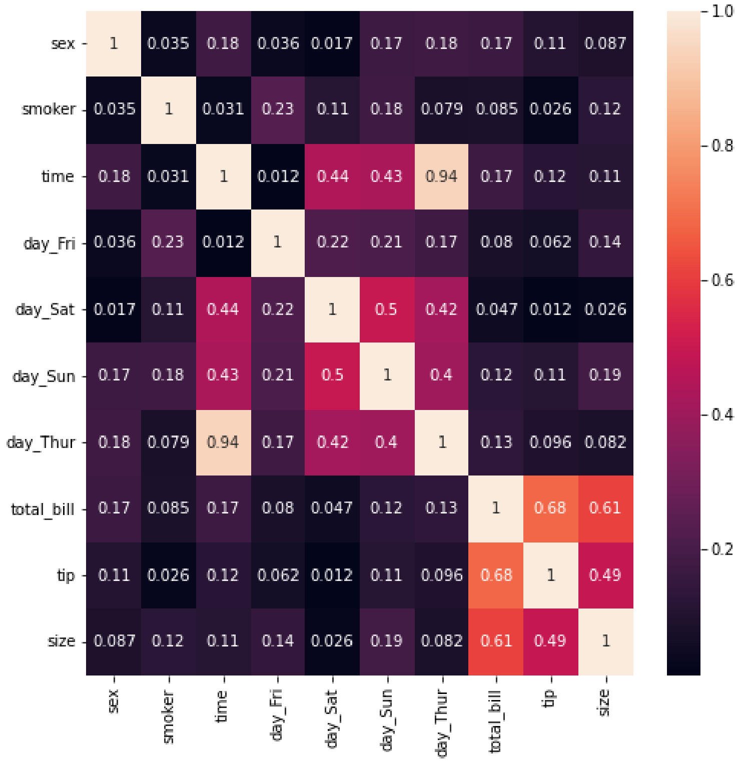
Lets see the latest version of the DataFrame.

In [69]:
```
df.head()
```

Out[69]:

	sex	smoker	time	day_Fri	day_Sat	day_Sun	day_Thur	total_bill	tip	size
0	0	0	0	0	0	1	0	0.291579	0.001111	0.2
1	1	0	0	0	0	1	0	0.152283	0.073333	0.4
2	1	0	0	0	0	1	0	0.375786	0.277778	0.4
3	1	0	0	0	0	1	0	0.431713	0.256667	0.2
4	0	0	0	0	0	1	0	0.450775	0.290000	0.6

In [70]:
```
correlation = df.corr().abs()
plt.figure(figsize=(8,8))
sns.heatmap(correlation, annot=True)
plt.show()
```



- ## Summarization:

There are two unique values in sex,time,smoker and four values in day column, so we have to encode sex,time,smoker with label encoder and encode day with one hot encoder.

Use statistical methods to understand the data and apply the required methods

```
In [3]: # Preview data
        df.head(10)
```

Out[3]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC Number	Payment ID
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325168603410	Sun2959
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071379779230	Sun4608
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011812112971322	Sun4458
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137647685994	Sun5260
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732618637221	Sun2251
5	25.29	4.71	Male	No	Sun	Dinner	4	6.32	Erik Smith	213140353657882	Sun9679
6	8.77	2.00	Male	No	Sun	Dinner	2	4.38	Kristopher Johnson	2223727524230344	Sun5985
7	26.88	3.12	Male	No	Sun	Dinner	4	6.72	Robert Buck	3514785077705092	Sun8157
8	15.04	1.96	Male	No	Sun	Dinner	2	7.52	Joseph Mcdonald	3522866365840377	Sun6820
9	14.78	3.23	Male	No	Sun	Dinner	2	7.39	Jerome Abbott	3532124519049786	Sun3775

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   total_bill        244 non-null    float64
 1   tip               244 non-null    float64
 2   sex               244 non-null    object
 3   smoker            244 non-null    object
 4   day               244 non-null    object
 5   time              244 non-null    object
 6   size              244 non-null    int64
 7   price_per_person  244 non-null    float64
 8   Payer Name        244 non-null    object
 9   CC Number         244 non-null    int64
 10  Payment ID        244 non-null    object
dtypes: float64(3), int64(2), object(6)
memory usage: 21.1+ KB
```

```
df.isnull().sum()
#finding if our data has null values or not
```

```
total_bill          0
tip                 0
sex                 0
smoker              0
day                 0
time                0
size                0
price_per_person    0
Payer Name          0
CC Number           0
Payment ID          0
dtype: int64
```

Dimensions of the dataset

```
# Dataset dimensions - (rows, columns)
df.shape
```

```
(244, 11)
```

Statistical summary of all attributes

```
df.describe()
```

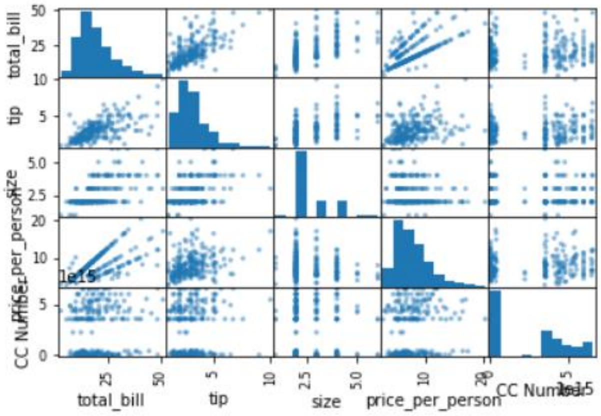
	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

- **Data Visualization:**

  We will visualize correlation with heatmap, use count plots to see if the women or men come to the restaurant more than one another. Then let's see if the tip left really depends on the gender of the customer with box plot.

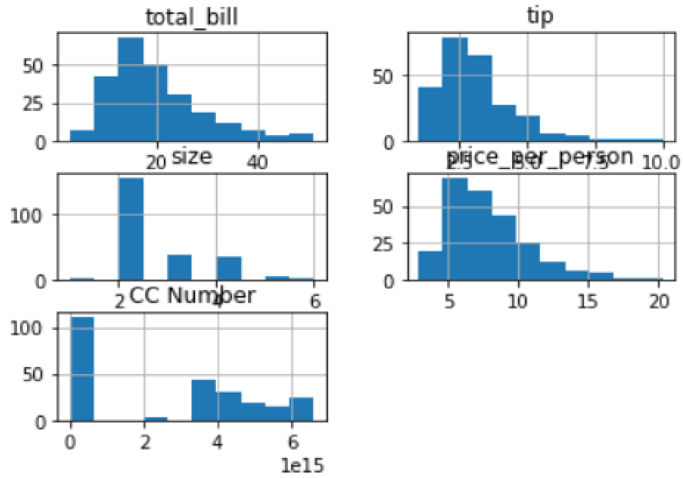
Scatter Matrix

```
from pandas.plotting import scatter_matrix
scatter_matrix(data)
plt.show()
```

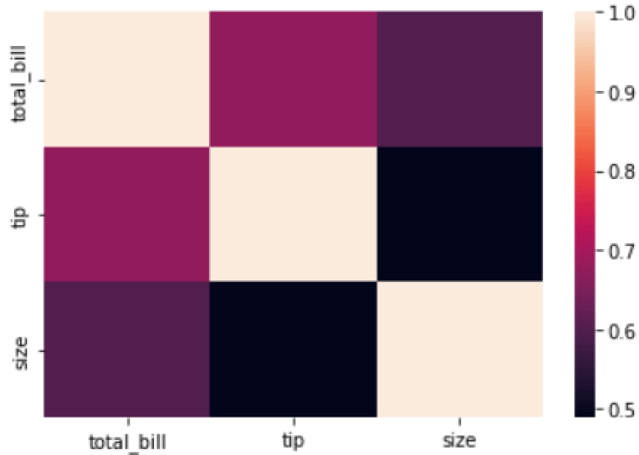


Histogram

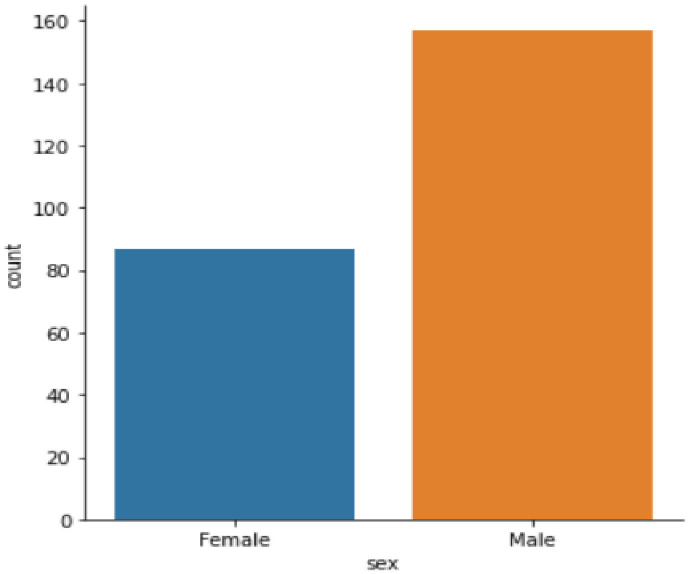
```
data.hist()
plt.show()
```



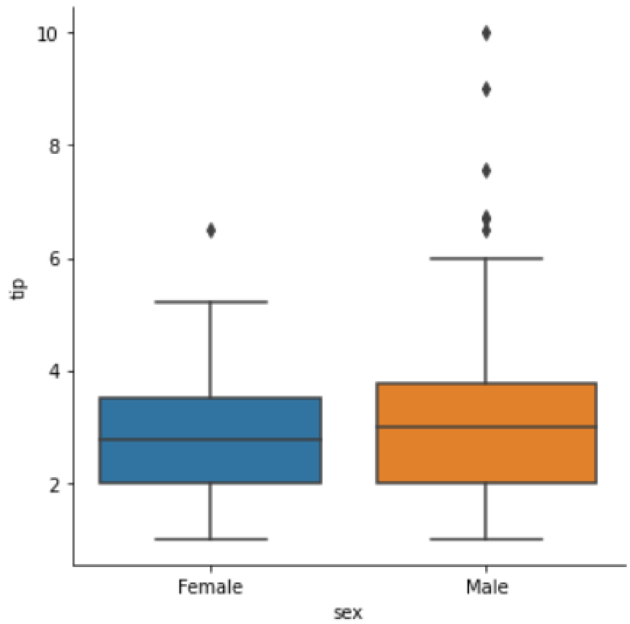

```
sns.heatmap(correlation)
plt.show()
```




```
sns.catplot(x="sex",data=df,kind="count")
plt.show()
```



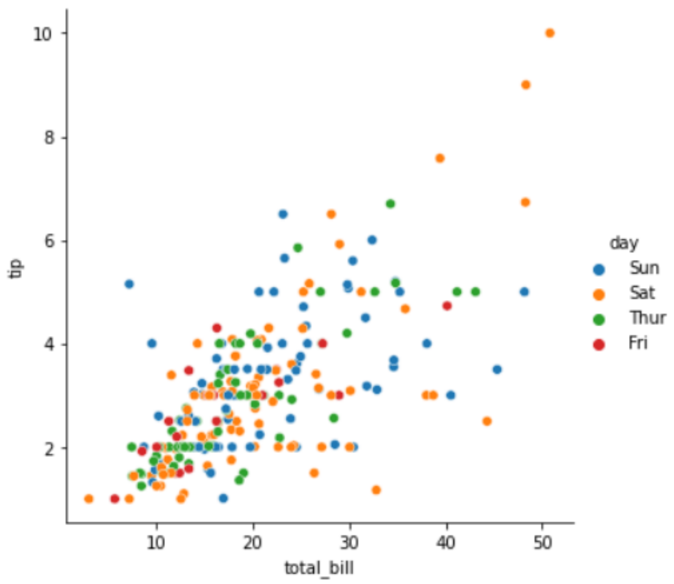
```
sns.catplot(x="sex",y="tip",data=df,kind="box")
plt.show()
```




```
df.corrwith(df["tip"])
```

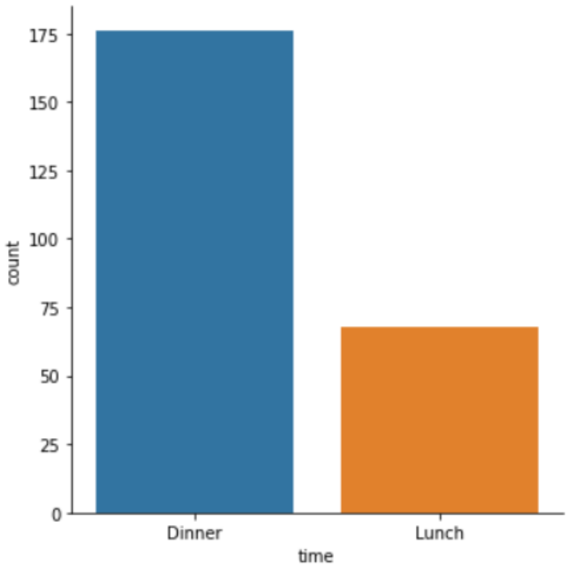
```
total_bill    0.675734
tip           1.000000
size          0.489299
dtype: float64
```

```
sns.relplot(x="total_bill",y="tip",data=df,kind="scatter",hue="day")
plt.show()
```



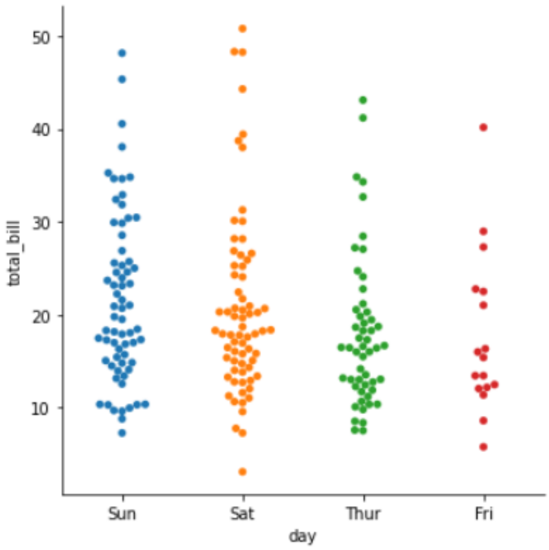
```
sns.catplot(x="time",data=df,kind="count")
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



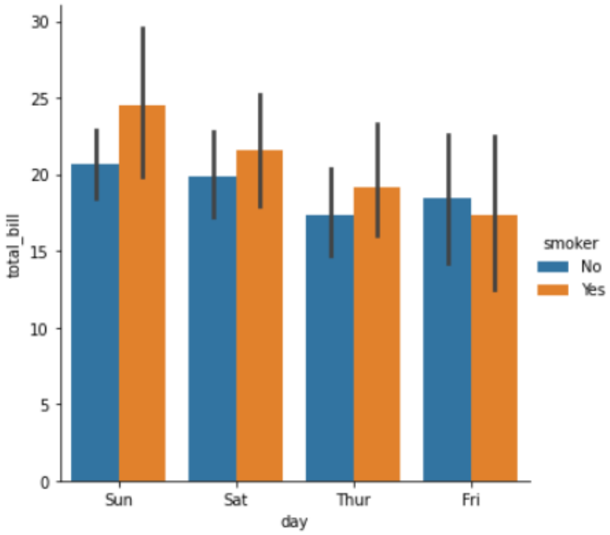
Data Visulization with total bill and day

```
sns.catplot(data=df, kind="swarm", x="day", y="total_bill")
plt.show()
```



Data Visulization with total bill,day with hue as smoker

```
sns.catplot(data=df, kind="bar", x="day", y="total_bill",hue="smoker")
plt.show()
```



Visualize the data using various plots like scatterplot, histograms, box plot, etc, and record your interpretations with varying values

4.Python packages

Brief on the python packages used for implementation of Machine learning algorithms pertaining to your project.

Pandas- Pandas is a popular Python library for data analysis. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for grouping, combining and filtering data.

numpy- NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions

matplotlib- Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots.,

Scikit-learn- is one of the most popular ML libraries for classical ML algorithms. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis.

seaborn, scatter_matrix, LabelEncoder, MinMaxScaler, LinearRegression, mean_absolute_error,mean_squared_error,r2_score, SVR, classification_report, GridSearchCV, KNeighborsRegressor

5.Learning Algorithms

1.Linear Regression

In [362]:
```
from sklearn.linear_model import LinearRegression
```

In [363]:
```
lr_model = LinearRegression()
lr_model.fit(X_train, Y_train)
y_pred_lr= regressor.predict(x_test)
y_pred_lr
```

Out[363]:
```
array([0.1875    , 0.2109375 , 0.19921875, 0.03515625, 0.28125   ,
       0.1171875 , 0.16015625, 0.234375  , 0.203125  , 0.38671875,
       0.2265625 , 0.23046875, 0.1484375 , 0.12890625, 0.2578125 ,
       0.3515625 , 0.08984375, 0.12890625, 0.12109375, 0.25390625,
       0.30078125, 0.19140625, 0.1640625 , 0.15625   , 0.140625  ,
       0.1640625 , 0.24609375, 0.3828125 , 0.30078125, 0.140625  ,
       0.1328125 , 0.12890625, 0.16015625, 0.0859375 , 0.1875    ,
       0.13671875, 0.17578125, 0.109375  , 0.50390625, 0.30859375,
       0.1875    , 0.12890625, 0.1640625 , 0.3671875 , 0.109375  ,
       0.1953125 , 0.16796875, 0.21875   , 0.17578125])
```

Coefficients (slope and intercept) of the model:

In [447]:
```
print("Coefficients : \n",lr_model.coef_)
```

```
Coefficients :
 [3.03431662e-03 6.14180904e-03 5.62730595e-02 2.27903836e+13
 2.27903836e+13 2.27903836e+13 2.27903836e+13 4.57299086e-01
 1.24060408e-01]
```

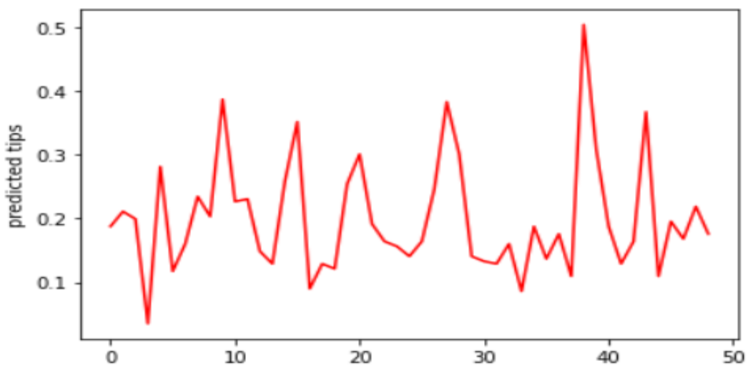
In [448]:
```
print("Intercept: \n",lr_model.intercept_)
```

```
Intercept:
 -22790383645430.094
```

In [407]:
```
mse_lr = mean_squared_error(y_test,y_pred_lr)
print("Mean Square Error from Linear Regression is ",mse_lr)
r2_lr = r2_score(y_test,y_pred_lr)
print("r2 score : ",r2_lr)
rmse_lr = sqrt(mse_lr)
print("Root Mean Square Error is : ",rmse_lr)
mae_lr = mean_absolute_error(y_test,y_pred_lr)
print("Mean absolute error : ",mae_lr)
```

```
Mean Square Error from Linear Regression is  0.0126962432915151
r2 score :  0.4874850783445437
Root Mean Square Error is :  0.11267760776443161
Mean absolute error :  0.08533907312925168
```

In [457]:
```
plt.plot(y_pred_lr,label="predictions",color="red")
plt.ylabel("predicted tips")
plt.show()
```



2.Support Vector Machine

In [368]:
```
from sklearn.svm import SVR
```

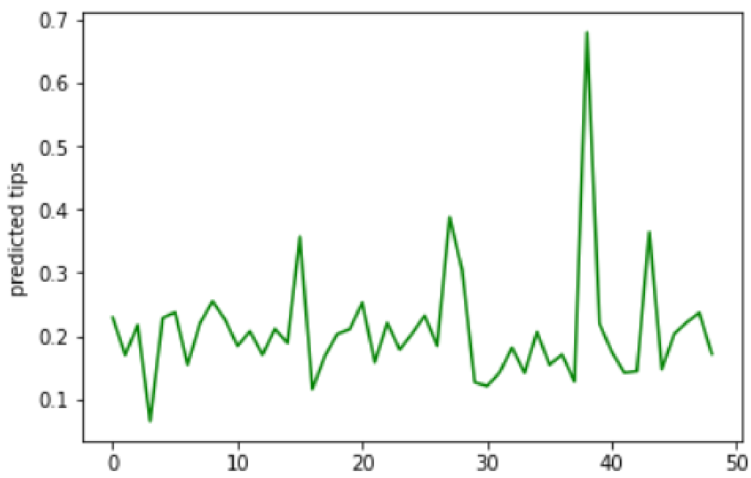
In [431]:
```
svm_model = SVR()
svm_model.fit(X_train,Y_train)
y_pred_svm=svm.predict(x_test)
y_pred_svm
```

Out[431]:
```
array([0.22912963, 0.16945861, 0.21730697, 0.06516993, 0.22840382,
       0.2373961 , 0.15440904, 0.21995004, 0.25469971, 0.22592163,
       0.18398536, 0.2073096 , 0.17013109, 0.21109027, 0.18877695,
       0.3568662 , 0.11557221, 0.16816226, 0.20301329, 0.210742  ,
       0.25273415, 0.15837395, 0.22059682, 0.17821509, 0.20393353,
       0.23221977, 0.18428754, 0.38763289, 0.30406179, 0.12676024,
       0.12084936, 0.14196887, 0.18142905, 0.14143621, 0.20647639,
       0.15418649, 0.17055729, 0.12781283, 0.67976172, 0.21961316,
       0.17447305, 0.14163009, 0.14416758, 0.36424492, 0.14713949,
       0.20335716, 0.22151108, 0.23657785, 0.17148229])
```

In [432]:
```
mse_svm=mean_squared_error(y_pred_svm,y_test)
print("Mean Square Error from SVM is ",mse_svm)
mae_svm = mean_absolute_error(y_pred_svm,y_test)
print("Mean absolute error from SVM is",mae_svm)
r2_svm = r2_score(y_test,y_pred_svm)
print("r2 score : ",r2_svm)
rmse_svm = sqrt(mse_svm)
print("Root Mean Square Error is : ",rmse_svm)
```

```
Mean Square Error from SVM is  0.011578283461957594
Mean absolute error from SVM is 0.08505877681723711
r2 score :  0.5326142619387593
Root Mean Square Error is :  0.10760243241654713
```

In [458]:
```
plt.plot(y_pred_svm,label="predictions",color="green")
plt.ylabel("predicted tips")
plt.show()
```



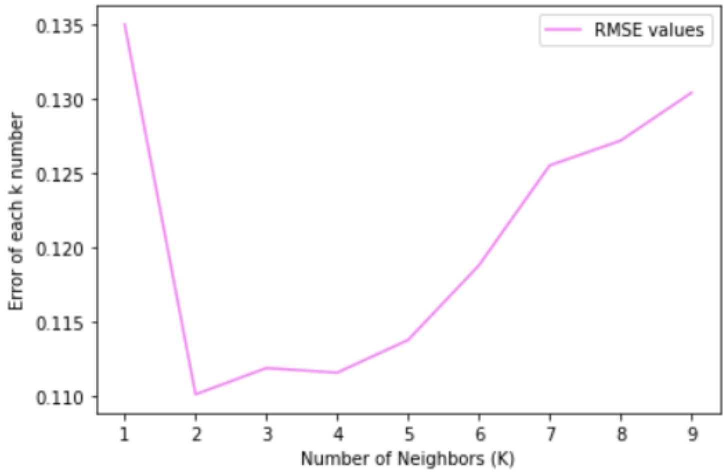
3.KNN

In [434]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
```

In [435]:
```
rmse_val = [] #to store rmse values for different k
kn = 10
nums=[]
for i in range(1,10):
    nums.append(i)
for K in nums:
    model = KNeighborsRegressor(n_neighbors = K)
    model.fit(X_train, Y_train)  #fit the model
    pred=model.predict(x_test) #make prediction on test set
    error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
    rmse_val.append(error) #store rmse values
print(rmse_val)
```

```
[0.1349982969760913, 0.11013742875333557, 0.11190615031597696, 0.111596833739
04246, 0.11378901765626871, 0.11880162709575606, 0.1255275036368599, 0.127187
0696640202, 0.1304071536059685]
```

In [459]:
```
plt.plot(range(1,kn),rmse_val,color="violet")
plt.legend(['RMSE values'])
plt.ylabel('Error of each k number')
plt.xlabel('Number of Neighbors (K)')
plt.tight_layout()
plt.show()
```



In [467]:
```
params = {'n_neighbors':[2,3,4,5,6,7,8,9]}
grid_search = GridSearchCV(KNeighborsRegressor(),params,cv=5)
grid_search.fit(X_train,Y_train)
grid_search.best_params_
```

Out[467]: {'n_neighbors': 4}

In [468]:
```
max_k_neighbor =Accuracy.argmax()
neigh = KNeighborsRegressor(max_k_neighbor+1)
neigh.fit(X_train,Y_train)
neigh
```

Out[468]: KNeighborsRegressor(n_neighbors=1)

4.Random Forest Regressor

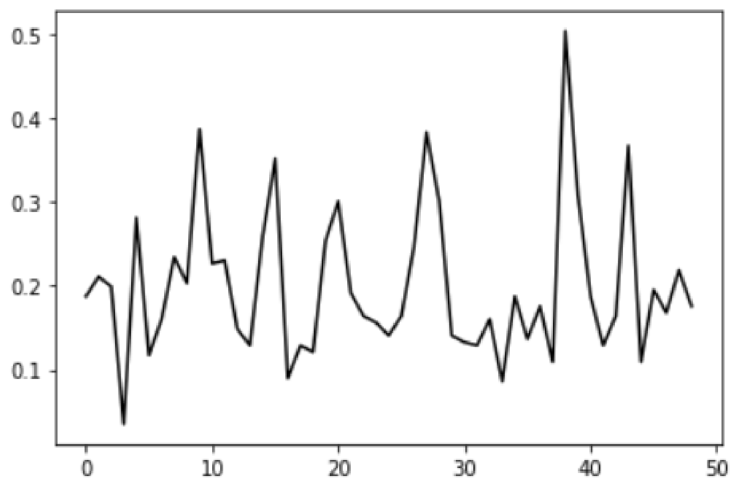
In [469]:
```
from sklearn.ensemble import RandomForestRegressor
RFR_model= RandomForestRegressor()
RFR_model.fit(X_train, Y_train)
y_pred_RFR= regressor.predict(x_test)
y_pred_RFR
```

Out[469]:
```
array([0.1875    , 0.2109375 , 0.19921875, 0.03515625, 0.28125   ,
       0.1171875 , 0.16015625, 0.234375  , 0.203125  , 0.38671875,
       0.2265625 , 0.23046875, 0.1484375 , 0.12890625, 0.2578125 ,
       0.3515625 , 0.08984375, 0.12890625, 0.12109375, 0.25390625,
       0.30078125, 0.19140625, 0.1640625 , 0.15625   , 0.140625  ,
       0.1640625 , 0.24609375, 0.3828125 , 0.30078125, 0.140625  ,
       0.1328125 , 0.12890625, 0.16015625, 0.0859375 , 0.1875    ,
       0.13671875, 0.17578125, 0.109375  , 0.50390625, 0.30859375,
       0.1875    , 0.12890625, 0.1640625 , 0.3671875 , 0.109375  ,
       0.1953125 , 0.16796875, 0.21875   , 0.17578125])
```

In [470]:
```
mse_RFR=mean_squared_error(y_pred_RFR,y_test)
print("Mean Square Error from RFR is ",mse_RFR)
mae_RFR = mean_absolute_error(y_pred_RFR,y_test)
print("Mean absolute error from RFR is",mae_RFR)
r2_RFR = r2_score(y_test,y_pred_RFR)
print("r2 score : ",r2_RFR)
rmse_RFR = sqrt(mse_RFR)
print("Root Mean Square Error is : ",rmse_RFR)
```

```
Mean Square Error from RFR is  0.0126962432915151
Mean absolute error from RFR is 0.08533907312925168
r2 score :  0.4874850783445437
Root Mean Square Error is :  0.11267760776443161
```

In [471]:
```
plt.plot(y_pred_RFR,label="predictions",color="black")
plt.show()
```



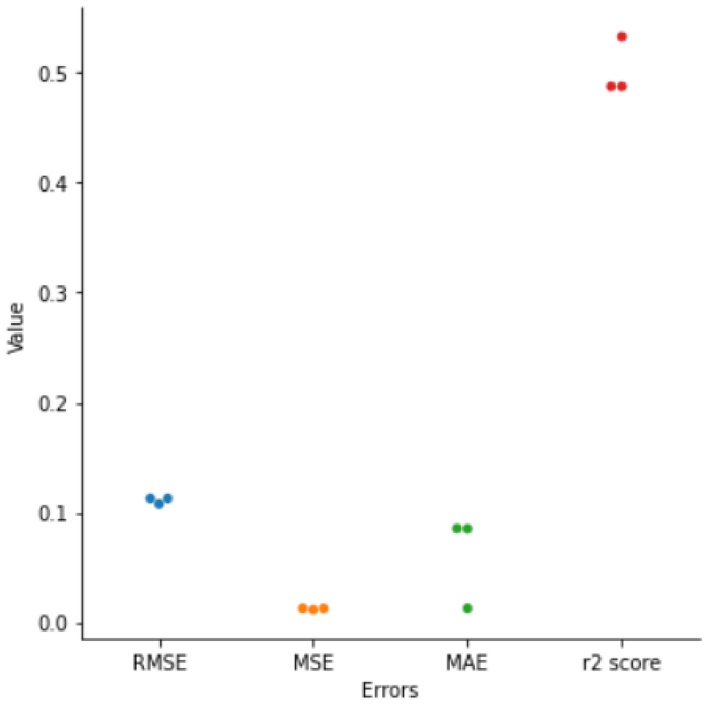
Comparing Models

In [472]:
```
data={
    "Model":["Linear Regression","SVM","Random Forest Regressor"],
    "RMSE":[rmse_lr,rmse_svm,rmse_RFR],
    "MSE":[mse_lr,mse_svm,mse_RFR],
    "MAE":[mae_lr,mae_svm,mse_RFR],
    "r2 score":[r2_lr,r2_svm,r2_RFR]
}
error_df = pd.DataFrame(data)
error_df
```

Out[472]:

	Model	RMSE	MSE	MAE	r2 score
0	Linear Regression	0.112678	0.012696	0.085339	0.487485
1	SVM	0.107602	0.011578	0.085059	0.532614
2	Random Forest Regressor	0.112678	0.012696	0.012696	0.487485

In [473]:
```
sns.catplot(data=error_df,kind="swarm")
plt.xlabel("Errors")
plt.ylabel("Value")
plt.show()
```



In [474]:
```
model_scores={"KNN":neigh.score(x_test,y_test),"SVM":svm.score(x_test,y_test),
model_scores
```

Out[474]:
```
{'KNN': 0.26432184927452496,
 'SVM': 0.5326142619387593,
 'Linear Regression': 0.5010935956419856,
 'Random Forest Regressor': 0.6082426923632401}
```

In [476]:
```
plt.figure(figsize=(15,5))
plt.bar(*zip(*model_scores.items()),color=["orange","grey","green","red"])
plt.xlabel("Models")
plt.ylabel("accuracy")
plt.show()
```