In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()

df = pd.read_csv('Taxifare.csv')
df.head()
```

Out[1]:

	unique_id	amount	date_time_of_pickup	longitude_of_pickup	latitude_of_pickup	longitude_of_dropoff	lat
0	26:21.0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	
1	52:16.0	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	
2	35:00.0	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	
3	30:42.0	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	
4	51:00.0	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	

In [2]:
```
df.shape
```

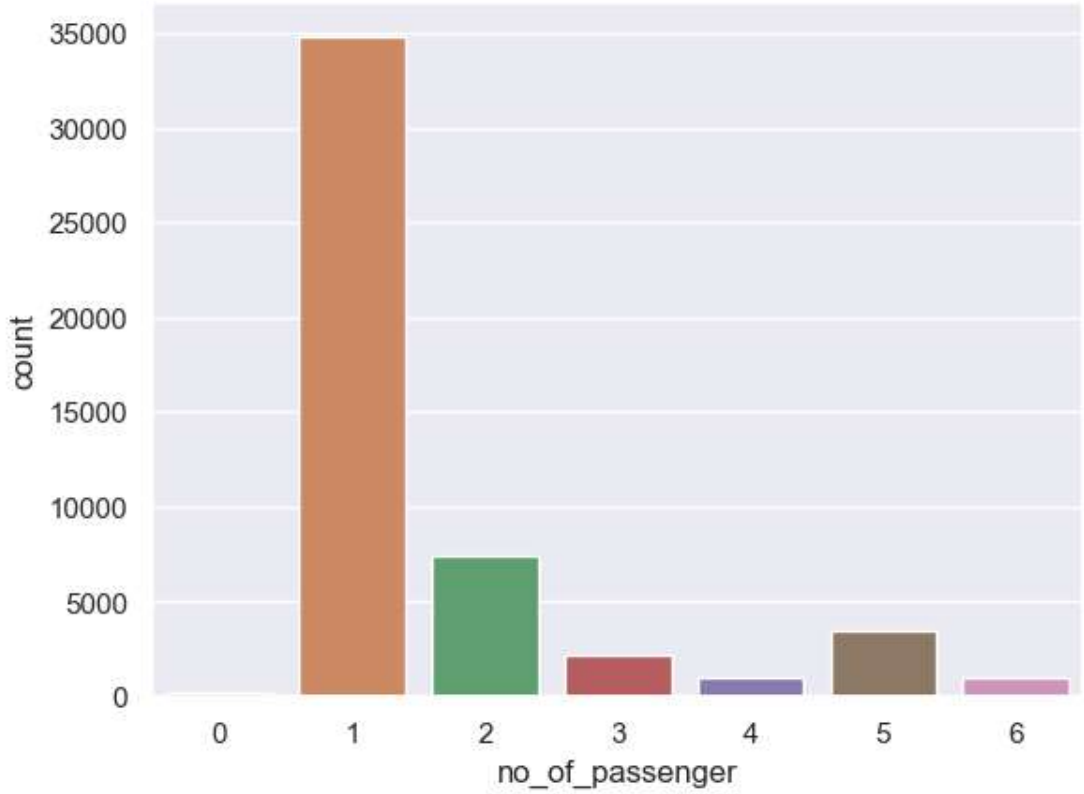
Out[2]:
```
(50000, 8)
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   unique_id            50000 non-null  object 
 1   amount               50000 non-null  float64
 2   date_time_of_pickup  50000 non-null  object 
 3   longitude_of_pickup  50000 non-null  float64
 4   latitude_of_pickup   50000 non-null  float64
 5   longitude_of_dropoff 50000 non-null  float64
 6   latitude_of_dropoff  50000 non-null  float64
 7   no_of_passenger      50000 non-null  int64  
dtypes: float64(5), int64(1), object(2)
memory usage: 3.1+ MB
```

In [4]:
```
sns.countplot(x=df['no_of_passenger'])
```

Out[4]:
```
<Axes: xlabel='no_of_passenger', ylabel='count'>
```



In [5]:
```
import pandas as pd
df = pd.read_csv('Taxifare.csv')
df = df[df['no_of_passenger'] == 1]
df = df.drop(['unique_id', 'no_of_passenger'], axis=1)
df.head()
```

Out[5]:

	amount	date_time_of_pickup	longitude_of_pickup	latitude_of_pickup	longitude_of_dropoff	latitude_of_dr
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.71
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.78
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.75
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.78
5	12.1	2011-01-06 09:50:45 UTC	-74.000964	40.731630	-73.972892	40.75

In [6]:
```
df.shape
```

Out[6]:
```
(34808, 6)
```

In [7]:
```
# importing the module
import numpy as np
# converting 'Field_2' from float to int
df['amount'] = df['amount'].apply(np.float64)

# displaying the datatypes
display(df.dtypes)
```

```
amount                  float64
date_time_of_pickup      object
longitude_of_pickup     float64
latitude_of_pickup      float64
longitude_of_dropoff    float64
latitude_of_dropoff     float64
dtype: object
```

In [8]:
```
import datetime
from math import sqrt

for i, row in df.iterrows():
    dt = datetime.datetime.strptime(row['date_time_of_pickup'], '%Y-%m-%d %H:%M:%S UTC'
    df.at[i, 'day_of_week'] = dt.weekday()
    df.at[i, 'pickup_time'] = dt.hour
    x = (row['longitude_of_dropoff'] - row['longitude_of_pickup']) * 54.6 # 1 degree ==
    y = (row['latitude_of_dropoff'] - row['latitude_of_pickup']) * 69.0   # 1 degree ==
    distance = sqrt(x**2 + y**2)
    df.at[i, 'distance'] = distance

df.head()
```

Out[8]:

	amount	date_time_of_pickup	longitude_of_pickup	latitude_of_pickup	longitude_of_dropoff	latitude_of_dr
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.71
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.78
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.75
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.78
5	12.1	2011-01-06 09:50:45 UTC	-74.000964	40.731630	-73.972892	40.75

In [9]:
```
df.drop(columns=['date_time_of_pickup', 'longitude_of_pickup', 'latitude_of_pickup', 'l
df.head()
```

Out[9]:

	amount	day_of_week	pickup_time	distance
0	4.5	0.0	17.0	0.641024
1	16.9	1.0	16.0	5.275538
3	7.7	5.0	4.0	1.738444
4	5.3	1.0	7.0	1.253707
5	12.1	3.0	9.0	2.391384

In [10]:
```
corr_matrix = df.corr()
corr_matrix["amount"].sort_values(ascending=False)
```

Out[10]:
```
amount         1.000000
distance       0.014725
day_of_week    0.010151
pickup_time   -0.015876
Name: amount, dtype: float64
```

In [11]:
```
df.describe()
```

Out[11]:

	amount	day_of_week	pickup_time	distance
count	34808.000000	34808.000000	34808.000000	34808.000000
mean	11.210226	2.947713	13.382757	11.850895
std	9.527580	1.942392	6.401627	246.753948
min	-5.000000	0.000000	0.000000	0.000000
25%	6.000000	1.000000	9.000000	0.771727
50%	8.500000	3.000000	14.000000	1.322725
75%	12.500000	5.000000	19.000000	2.414889
max	200.000000	6.000000	23.000000	24861.003946

In [12]:
```
df = df[(df['distance'] > 1.0) & (df['distance'] < 10.0)]
df = df[(df['amount'] > 0.0) & (df['amount'] < 50.0)]
df.shape
```

Out[12]:
```
(21318, 4)
```

In [13]:
```
corr_matrix = df.corr()
corr_matrix["amount"].sort_values(ascending=False)
```

Out[13]:
```
amount         1.000000
distance       0.848715
day_of_week    0.002447
pickup_time   -0.018968
Name: amount, dtype: float64
```

That looks better! The correlation between the day of the week, the hour of the day, and fare amount is still weak, but let's leave those columns in there since it makes sense that it might take longer to get from point A to point B during rush hour, or that traffic at 5:00 p.m. Friday might be different than traffic at 5:00 p.m. on Saturday.

Train a regression model Now it's time build a regression model and train it with the data prepared in the previous exercise. We'll try three different regression algorithms to determine which one produces the most accurate results, and use cross-validation to increase our confidence in those results. Start by splitting the data for training and testing.

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

x = df.drop(['amount'], axis=1)
y = df['amount']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0
```

Train a linear-regression model

In [15]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(x_train, y_train)
```

Out[15]:
```
LinearRegression
LinearRegression()
```

What is the model's R-squared score?

In [16]:
```
model.score(x_test, y_test)
```

Out[16]:
```
0.7294968092631228
```

Score the model again using 5-fold cross-validation

In [17]:
```
from sklearn.model_selection import cross_val_score

cross_val_score(model, x, y, cv=5).mean()
```

Out[17]:
```
0.7207057353445661
```

Measure the model's mean absolute error (MAE)

In [18]:
```
from sklearn.metrics import mean_absolute_error

mean_absolute_error(y_test, model.predict(x_test))
```

Out[18]:
```
2.428339345293518
```

Now train a RandomForestRegressor using the same dataset and see how its accuracy compares

In [25]:
```
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import GradientBoostingRegressor

# Assuming x and y are your feature matrix and target variable
# Replace this with your actual data loading/preparation steps
# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0

# Create the GradientBoostingRegressor model
model = GradientBoostingRegressor(random_state=0)

# Fit the model on the training data
model.fit(x_train, y_train)

# Use cross_val_score to evaluate the model
cross_val_mean = cross_val_score(model, x, y, cv=5).mean()

# Print the mean cross-validated score
print("Mean Cross-Validated Score:", cross_val_mean)
```

```
Mean Cross-Validated Score: 0.7424626872254264
```

In [27]:
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(random_state=0)
model.fit(x_train, y_train)

cross_val_score(model, x, y, cv=5).mean()
```

Out[27]:
```
0.6997168858484374
```

Which model produced the highest cross-validated coefficient of determination?

Use the model to predict fare amounts Finish up by using the trained model to make a pair of predictions. First, estimate what it will cost to hire a taxi for a 2-mile trip at 5:00 p.m. on Friday afternoon

In [28]:
```
model.predict([[4, 17, 2.0]])
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does no
t have valid feature names, but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

Out[28]:
```
array([16.612])
```

Now predict the fare amount for a 2-mile trip taken at 5:00 p.m. one day later (on Saturday).

In [29]:
```
model.predict([[5, 17, 2.0]])
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does no
t have valid feature names, but RandomForestRegressor was fitted with feature names
  warnings.warn(
```

Out[29]:
```
array([13.478])
```

In [ ]:
```

```