In [1]:
```
import pandas as pd
from plotnine import *
import math
import patchworklib as pw
import matplotlib.pyplot as plt

from IPython.display import set_matplotlib_formats
set_matplotlib_formats('svg')
```

```
cannot import name '_draw_figure' from 'seaborn.utils' (/Users/chetanmunugala/
anaconda3/lib/python3.8/site-packages/seaborn/utils.py)
<Figure size 100x100 with 0 Axes>
```

In [2]:
```
train_df = pd.read_csv('train.csv',nrows= 100000)
```

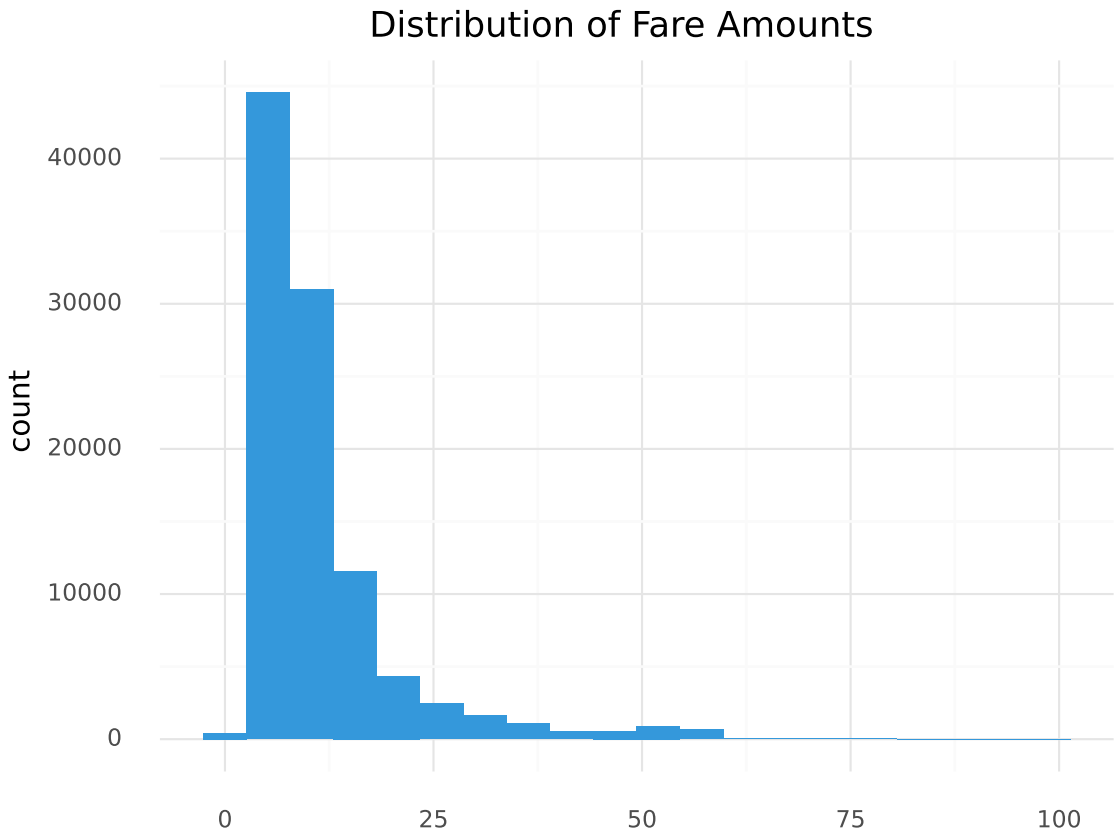
In [3]:
```
train_df.head()
```

Out[3]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropo
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	

In [4]:
```
fare_dist = list(train_df[(train_df.fare_amount < 100) & (train_df.fare_amount
```

In [5]:
```
p1 = ggplot() + aes(x=fare_dist) + geom_histogram(fill='#3498db',bins=20) + the
labs(title = 'Distribution of Fare Amounts')
p1
```



Out[5]: <ggplot: (8781979257079)>

Potentially Useful Features

1. Morning, Afternoon, Night
2. Distance
3. Passenger Count
4. Season

Feature Engineering

In [6]:
```
def return_season(month):
    month = int(month)
    if(month in [12,1,2]):
        return('Winter')
    elif(month in [3,4,5]):
        return('Spring')
    elif(month in [6,7,8]):
        return('Summer')
    elif(month in [9,10,11]):
        return('Fall')
    else:
        return('NA')
```

In [7]:
```
def return_time_of_day(hour):
    hour = int(hour)
    if(hour >= 13 and hour <= 18):
        return('Afternoon')
    elif(hour >= 19 or hour <= 5):
```

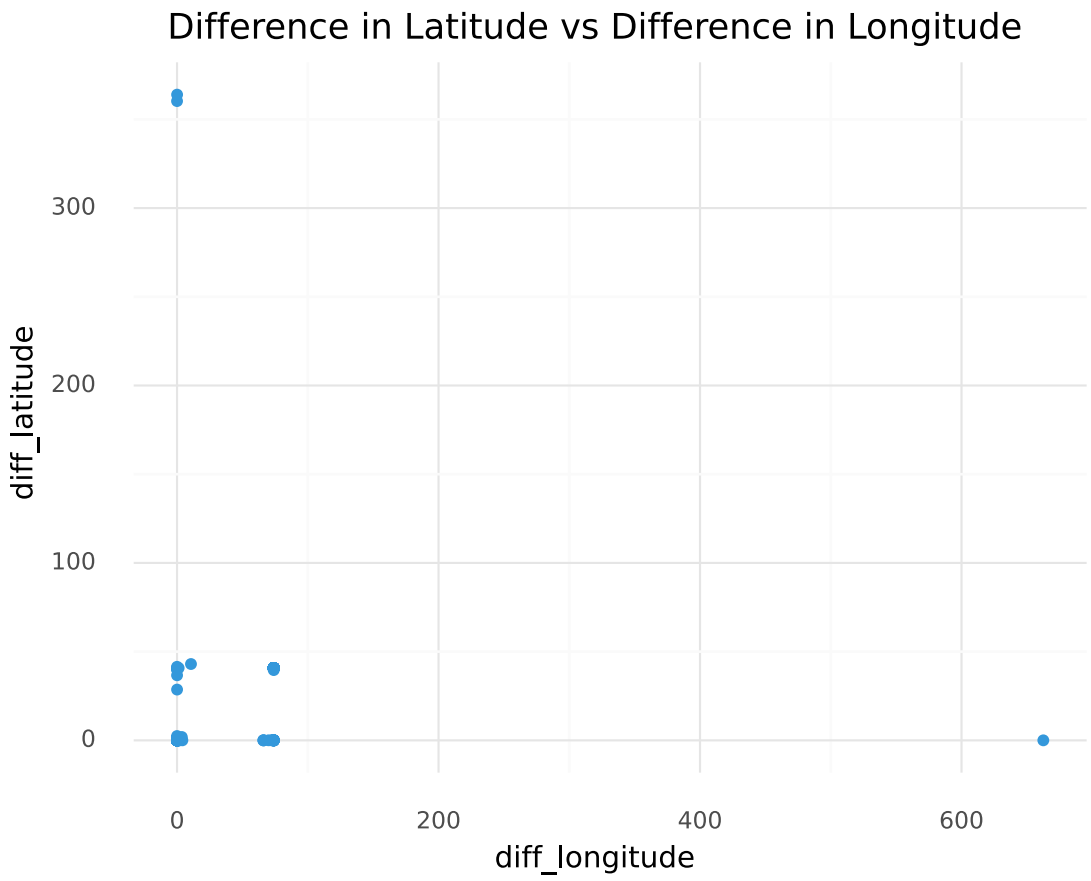
```
        return('Night')
    elif(hour>=6 and hour <= 12):
        return('Morning')
    else:
        return('NA')
```

In [8]:
```
def add_features(df):

    df['time'] = df['pickup_datetime'].apply(lambda x:x.split(' ')[1])
    df['date'] = df['pickup_datetime'].apply(lambda x:x.split(' ')[0])

    df['diff_longitude'] = (df['pickup_longitude']-df['dropoff_longitude']).abs
    df['diff_latitude'] = (df['pickup_latitude']-df['dropoff_latitude']).abs()

    df['hour'] = df['time'].apply(lambda x:x.split(':')[0])
    df['month'] = df['date'].apply(lambda x:int(x.split('-')[1]))

    df['time_of_day'] = df['hour'].apply(lambda x:return_time_of_day(x))
    df['season'] = df['month'].apply(lambda x:return_season(x))

    tod_dummies = pd.get_dummies(df['time_of_day'])
    season_dummies = pd.get_dummies(df['season'])

    df = pd.concat([df,tod_dummies,season_dummies],axis=1)

    return df
```

In [9]:
```
train_df = add_features(train_df)
```

Exploratory Data Analysis

In [10]:
```
outlier_plot = ggplot(train_df) + aes(x='diff_longitude',y='diff_latitude') + g
theme_minimal() + labs(title = 'Difference in Latitude vs Difference in Longitu
outlier_plot
```



Out[10]: <ggplot: (8781996794745)>

In [11]:
```
train_df = train_df[train_df.diff_longitude < 10]
train_df = train_df[train_df.diff_latitude < 10]
```

In [12]:
```
p2 = ggplot(train_df) + aes(x='passenger_count') + geom_histogram(fill='#E74C3C
labs(title = 'Distribution of Passenger Counts')
```

In [13]:
```
p3 = ggplot(train_df) + aes(x='diff_longitude') + geom_histogram(fill = '#3498d
theme_minimal()

p4 = ggplot(train_df) + aes(x='diff_latitude') + geom_histogram(fill = '#3498db
theme_minimal()
```

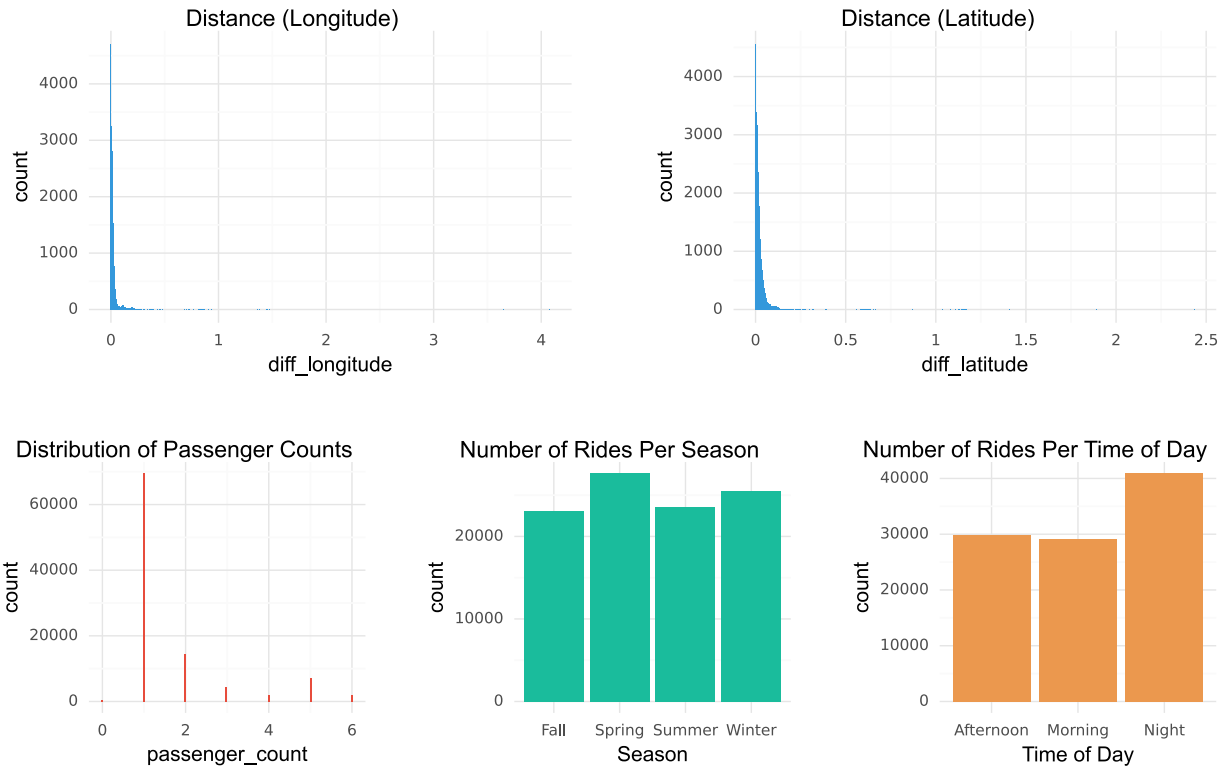
In [14]:
```
p5 = ggplot(train_df) + aes(x='factor(season)') + geom_bar(fill = '#1ABC9C') +
labs(title='Number of Rides Per Season',x='Season')

p6 = ggplot(train_df) + aes(x='factor(time_of_day)') + geom_bar(fill = '#EB984E
labs(title='Number of Rides Per Time of Day',x='Time of Day')
```

In [15]:
```
p2 = pw.load_ggplot(p2,figsize=(2.2,2))
p3 = pw.load_ggplot(p3,figsize=(3.3,2))
p4 = pw.load_ggplot(p4,figsize=(3.3,2))
p5 = pw.load_ggplot(p5,figsize=(2.2,2))
p6 = pw.load_ggplot(p6,figsize=(2.2,2))
```

In [16]:
```
final_plot = (p3 + p4)/(p2 + p5 + p6)
```

In [17]:
```
final_plot.savefig()
```

Out[17]:



Modeling

In [18]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

In [19]:
```
train_df = train_df[['passenger_count','diff_longitude','diff_latitude','Mornir
```

In [20]:
```
X_train = torch.tensor(train_df.iloc[:,:-1].values,dtype=torch.float32)
y_train = torch.tensor(train_df[['fare_amount']].values,dtype=torch.float32)
```

In [21]:
```
#device config
#device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [22]:
```
#hyperparameters
n_input = 10
n_hidden = 5
n_out = 1
num_epochs = 2
batch_size = 100
learning_rate = 0.3
```

In [23]:
```
model = nn.Sequential(nn.Linear(n_input, n_hidden),
                      nn.ReLU(),
                      nn.Linear(n_hidden, n_out))
```

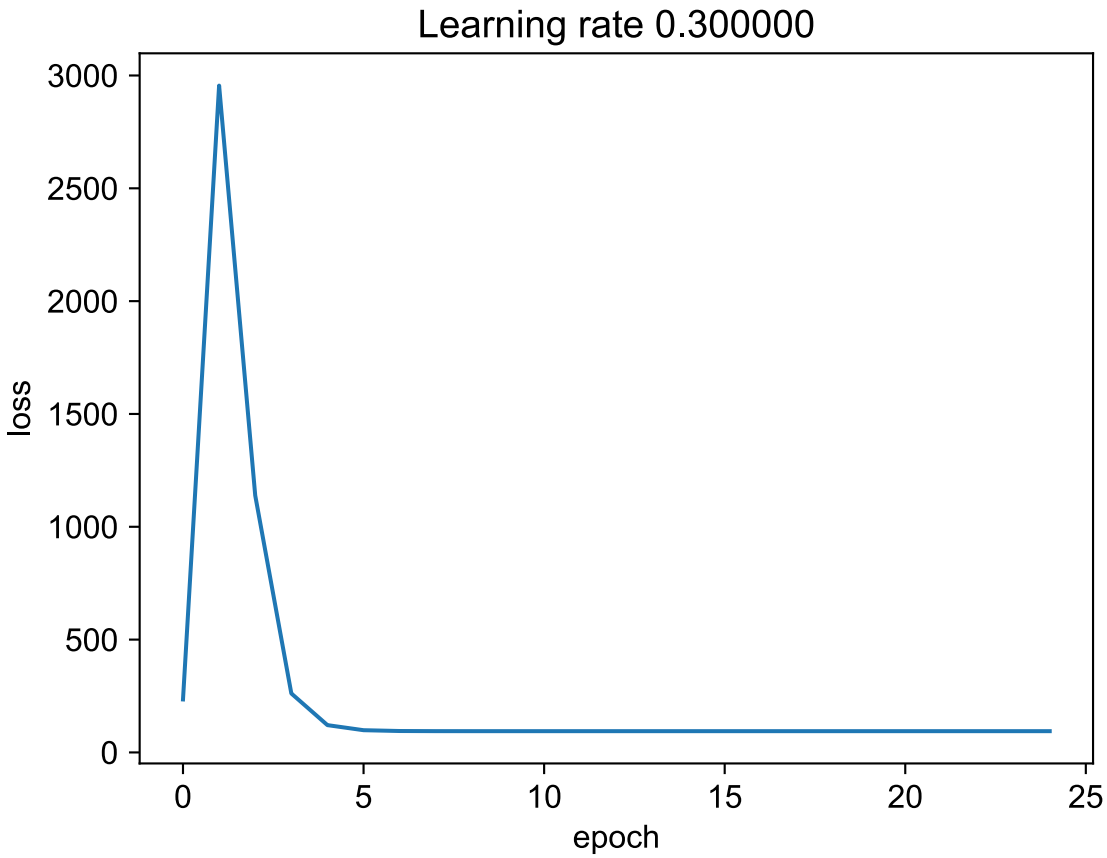
In [24]:
```
print(model)
```

```
Sequential(
  (0): Linear(in_features=10, out_features=5, bias=True)
  (1): ReLU()
  (2): Linear(in_features=5, out_features=1, bias=True)
)
```

In [25]:
```
loss_function = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

In [26]:
```
losses = []
for epoch in range(25):
    pred_y = model(X_train)
    loss = loss_function(pred_y, y_train)
    losses.append(loss.item())

    model.zero_grad()
    loss.backward()

    optimizer.step()
```

In [27]:
```
import matplotlib.pyplot as plt
plt.plot(losses)
plt.ylabel('loss')
plt.xlabel('epoch')
plt.title("Learning rate %f"%(learning_rate))
plt.show()
```



In [ ]: