In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
sns.set_style('whitegrid')
```

TRAIN DATA

In [3]:
```
df_train= pd.read_csv('train(2).csv',nrows=200000,parse_dates=['pickup_datetime']
```

EXPLORATORY DATA ANALYSIS

In [4]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 8 columns):
 #   Column             Non-Null Count   Dtype              
---  ------             --------------   -----              
 0   key                200000 non-null  object             
 1   fare_amount        200000 non-null  float64            
 2   pickup_datetime    200000 non-null  datetime64[ns, UTC]
 3   pickup_longitude   200000 non-null  float64            
 4   pickup_latitude    200000 non-null  float64            
 5   dropoff_longitude  199999 non-null  float64            
 6   dropoff_latitude   199999 non-null  float64            
 7   passenger_count    200000 non-null  int64              
dtypes: datetime64[ns, UTC](1), float64(5), int64(1), object(1)
memory usage: 12.2+ MB
```

In [5]:
```
df_train.describe()
```

Out[5]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passeng
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	20000
mean	11.342877	-72.506121	39.922326	-72.518673	39.925579	
std	9.837855	11.608097	10.048947	10.724226	6.751120	
min	-44.900000	-736.550000	-3116.285383	-1251.195890	-1189.615440	
25%	6.000000	-73.992050	40.735007	-73.991295	40.734092	
50%	8.500000	-73.981743	40.752761	-73.980072	40.753225	
75%	12.500000	-73.967068	40.767127	-73.963508	40.768070	
max	500.000000	2140.601160	1703.092772	40.851027	404.616667	

In [6]:
```
df_train.dtypes
```

Out[6]:
```
key                                object
fare_amount                       float64
pickup_datetime      datetime64[ns, UTC]
pickup_longitude                  float64
pickup_latitude                   float64
dropoff_longitude                 float64
dropoff_latitude                  float64
passenger_count                     int64
dtype: object
```

DATA PRE-PROCESSING

minimum fare-amount is negative,so we have to remove these observations

In [7]:
```
df_train = df_train[df_train['fare_amount'] >=0]
```

In [8]:
```
len(df_train)
```

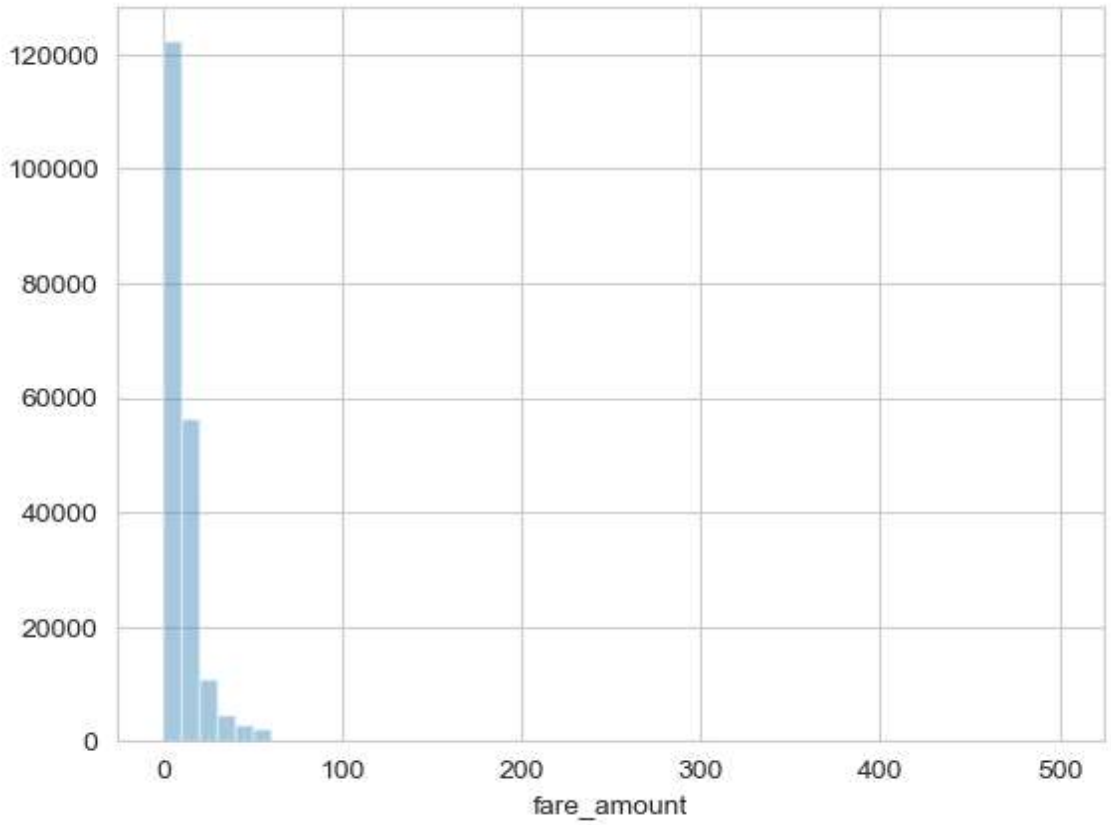
Out[8]:
```
199987
```

HISTOGRAM OF FARE AMOUNT

In [9]:
```
sns.distplot(df_train['fare_amount'],kde=False)
```

```
C:\Users\shiva\AppData\Local\Temp\ipykernel_16432\2875287990.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df_train['fare_amount'],kde=False)
```

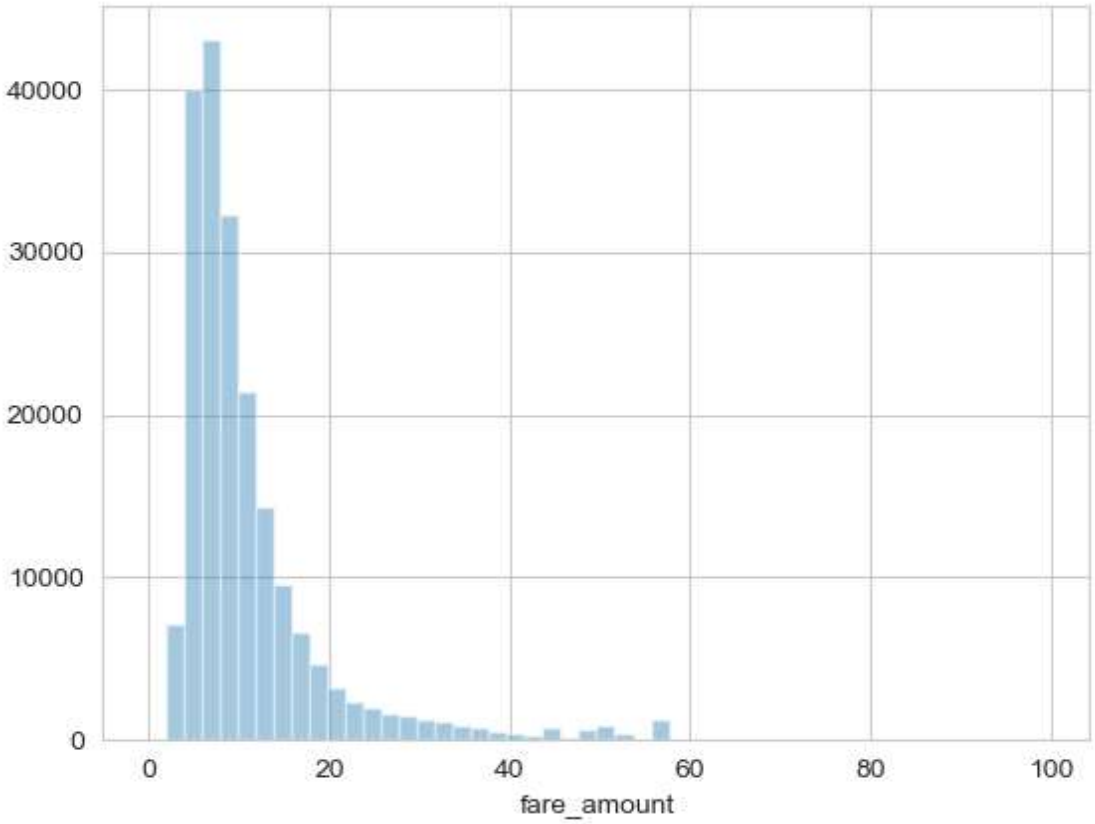
Out[9]:
```
<Axes: xlabel='fare_amount'>
```



In [10]:
```
sns.distplot(df_train[df_train['fare_amount'] < 100]['fare_amount'],kde=False)
```

```
C:\Users\shiva\AppData\Local\Temp\ipykernel_16432\4058103289.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df_train[df_train['fare_amount'] < 100]['fare_amount'],kde=False)
```

Out[10]:
```
<Axes: xlabel='fare_amount'>
```



NOTICE - small spikes between 40 and 60

REMOVE MISSING DATA

In [11]:
```
print(df_train.isnull().sum())
```

```
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

In [12]:
```
df_train = df_train.dropna(how='any',axis='rows')
```

In [13]:
```
len(df_train)
```

Out[13]:
```
199986
```

TEST DATA

In [14]:
```
df_test = pd.read_csv('test(2).csv')
```

In [15]:
```
df_test.head(5)
```

Out[15]:

	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pa
0	08:24.0	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	
1	08:24.0	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	
2	53:44.0	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	
3	12:12.0	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	
4	12:12.0	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	

In [16]:
```
df_test.describe().T
```

Out[16]:

	count	mean	std	min	25%	50%	75%	
pickup_longitude	9914.0	-73.974722	0.042774	-74.252193	-73.992501	-73.982326	-73.968013	-72.98
pickup_latitude	9914.0	40.751041	0.033541	40.573143	40.736125	40.753051	40.767113	41.70
dropoff_longitude	9914.0	-73.973657	0.039072	-74.263242	-73.991247	-73.980015	-73.964059	-72.99
dropoff_latitude	9914.0	40.751743	0.035435	40.568973	40.735254	40.754065	40.768757	41.69
passenger_count	9914.0	1.671273	1.278747	1.000000	1.000000	1.000000	2.000000	6.00

NewYork City coordinates are https://www.travelmath.com/cities/New+York,+NY (https://www.travelmath.com/cities/New+York,+NY)

In [17]:
```
latitude = 40.7141667
longitude= -74.0063889
```

Here, we will try to define bounding box of intrest by[long_max.latt_min,latt_max] using the minimum and maximum coordinates from the test.csv

This will help us to make sure to train a model for thee full pickup/dropoff co-ordinate range for the test set

minimum and maximum longitude test set

In [18]:
```
print(min(df_test['pickup_longitude'].min(),df_test['dropoff_longitude'].min()))
```

```
-74.263242
```

In [19]:
```
print(max(df_test['pickup_longitude'].max(),df_test['dropoff_longitude'].max()))
```

```
-72.986532
```

In [20]:
```
print(min(df_train['pickup_longitude'].min(),df_train['dropoff_longitude'].min())
```

```
-1251.19589
```

In [21]:
```
print(max(df_train['pickup_longitude'].max(),df_train['dropoff_longitude'].max())
```

```
2140.60116
```

minimum and maximum latitude test set

In [22]:
```
print(min(df_test['pickup_latitude'].min(),df_test['dropoff_latitude'].min()))
```

```
40.568973
```

In [23]:
```
print(max(df_test['pickup_latitude'].max(),df_test['dropoff_latitude'].max()))
```

```
41.709555
```

In [24]:
```
print(min(df_train['pickup_latitude'].min(),df_train['dropoff_latitude'].min()))
```

```
-3116.285383
```

In [25]:
```
print(max(df_train['pickup_latitude'].max(),df_train['dropoff_latitude'].max()))
```

```
1703.092772
```

Function for selecting the bounding box

In [26]:
```
def select_within_boundingbox(df, BB):
    return ((df["pickup_longitude"] >= BB[0]) & (df["pickup_longitude"] <= BB[1])
            (df["pickup_latitude"] >= BB[2]) & (df["pickup_latitude"] <= BB[3]) &
            (df["dropoff_longitude"] >= BB[0]) & (df["dropoff_longitude"] <= BB[1
            (df["dropoff_latitude"] >= BB[2]) & (df["dropoff_latitude"] <= BB[3])
```

Load image of NYC Map for visualization

In [27]:
```
BB = (-74.3,-73.0,40.6,41.7)
```

In [28]:
```
nyc_map = plt.imread('image.png')
```

In [29]:
```
nyc_map.shape
```

Out[29]:
```
(1262, 1242, 3)
```

In [30]:
```
df_train=df_train[select_within_boundingbox(df_train,BB)]
```

In [31]:
```
print("New size {}".format(len(df_train)))
```

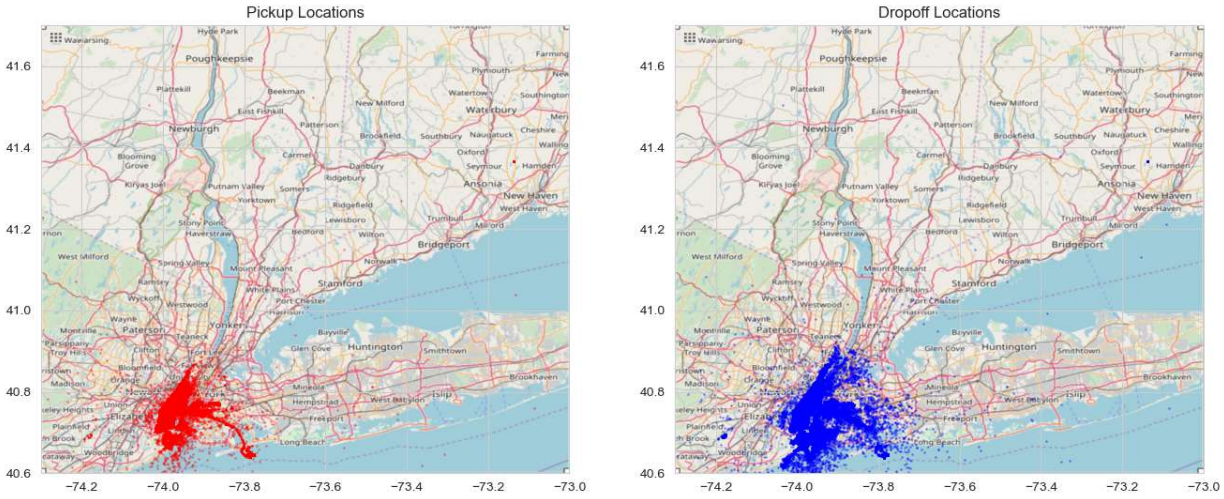
```
New size 195612
```

Function will be used for plotting data on NYC Map

In [32]:
```
def plot_on_map(df, BB, nyc_map, s=10, alpha=0.2):
    fig, axs = plt.subplots(1,2,figsize=(16, 10))
    axs[0].scatter(df["pickup_longitude"], df["pickup_latitude"], alpha = alpha,
    axs[0].set_xlim((BB[0], BB[1]))
    axs[0].set_ylim((BB[2], BB[3]))
    axs[0].set_title('Pickup Locations')
    axs[0].imshow(nyc_map, extent=BB)

    axs[1].scatter(df["dropoff_longitude"], df["dropoff_latitude"] , alpha = alph
    axs[1].set_xlim((BB[0], BB[1]))
    axs[1].set_ylim((BB[2], BB[3]))
    axs[1].set_title('Dropoff Locations')
    axs[1].imshow(nyc_map, extent=BB)
```

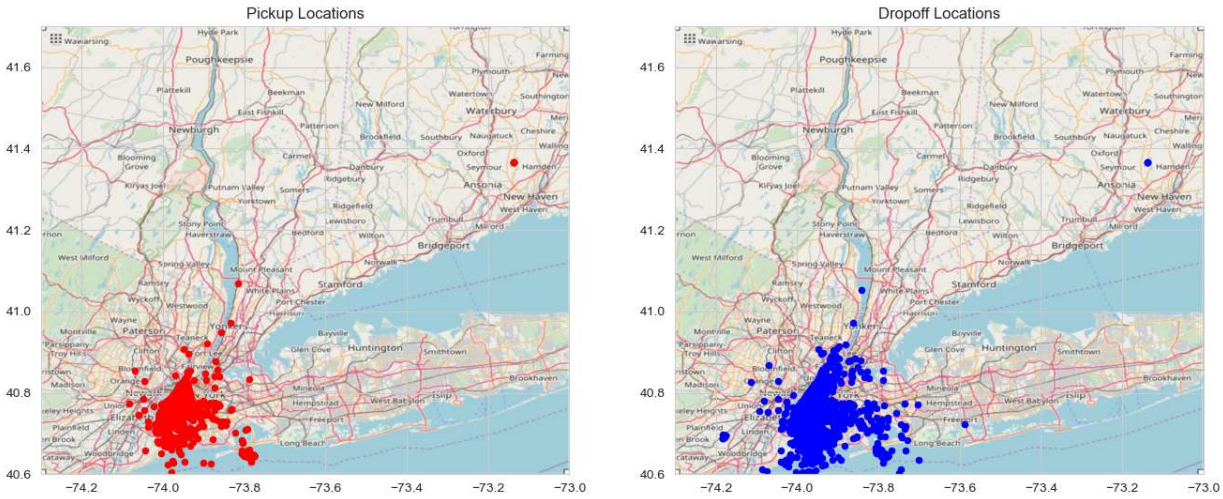
Plotting Training Data on map

In [33]:
```
plot_on_map(df_train, BB, nyc_map, s=1, alpha=0.3)
```



Plotting testing Data on Map

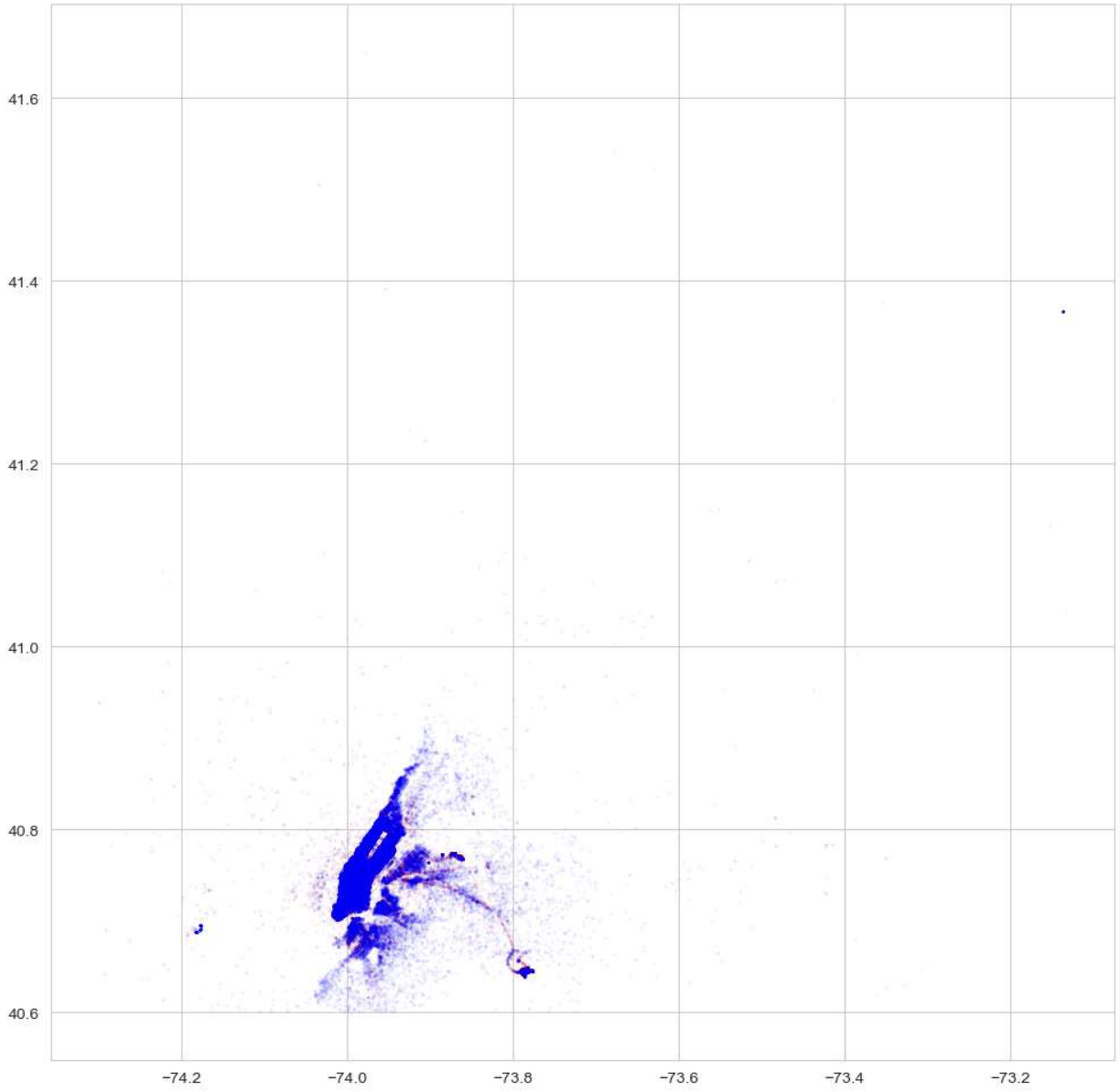
In [34]:
```
plot_on_map(df_test, BB, nyc_map, alpha=1.0, s=20)
```

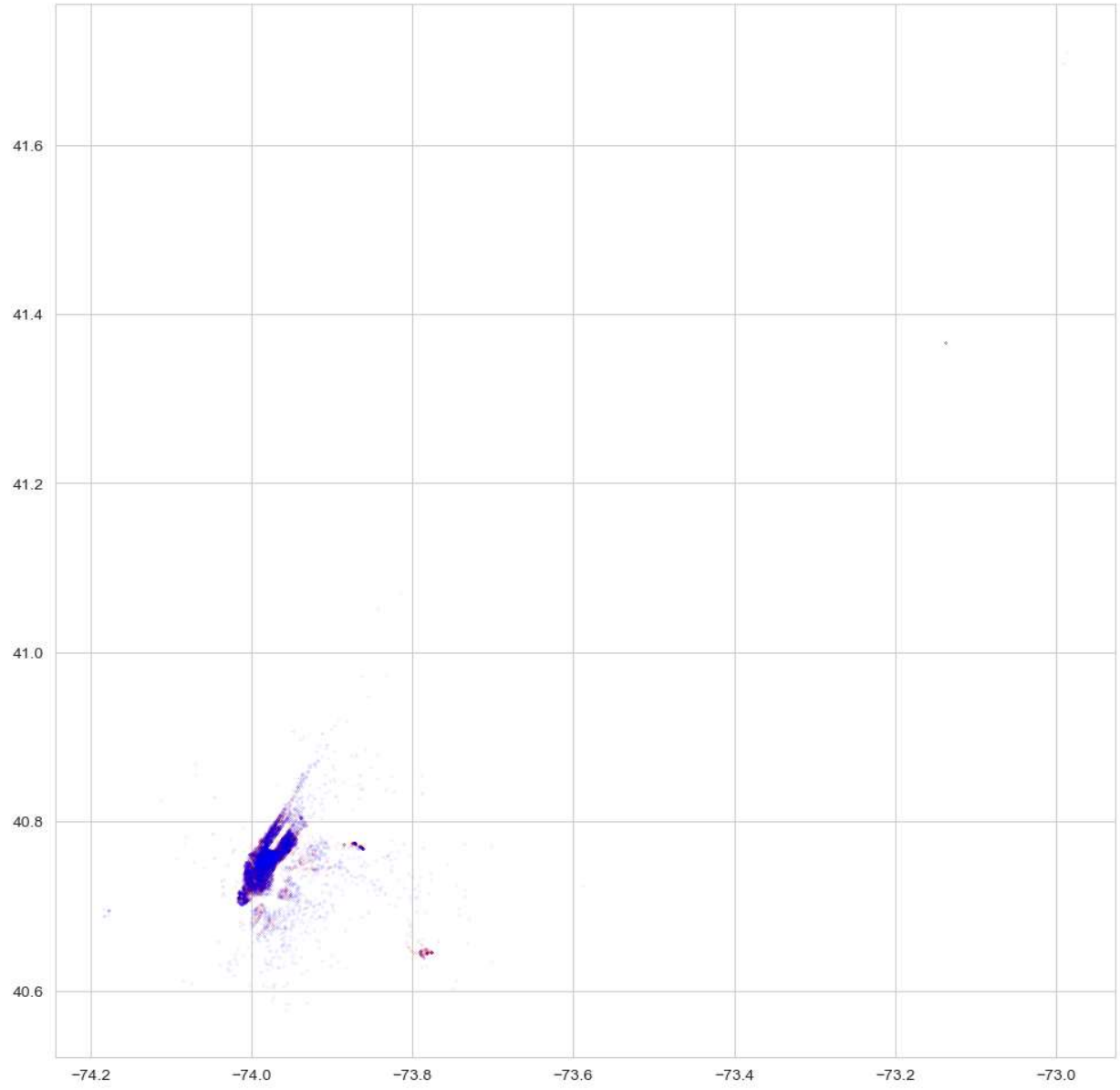


Scatter Plot

In [35]:
```
def plot_hires(df, BB, figsize=(12, 12), ax=None, c=('r', 'b')):
    if ax == None:
        fig, ax = plt.subplots(1, 1, figsize=figsize)

    idx = select_within_boundingbox(df, BB)
    ax.scatter(df[idx].pickup_longitude, df[idx].pickup_latitude, c=c[0], s=0.01,
    ax.scatter(df[idx].dropoff_longitude, df[idx].dropoff_latitude, c=c[1], s=0.0
```

In [36]:
```
plot_hires(df_train, (-74.3, -73.1, 40.6, 41.65))
plot_hires(df_test, (-74.26, -72.98, 40.57, 41.71))
```





In [37]:
```
# add time information
df_train['year'] = df_train["pickup_datetime"].apply(lambda t: t.year)
df_train['weekday'] = df_train["pickup_datetime"].apply(lambda t: t.weekday())
df_train['hour'] = df_train["pickup_datetime"].apply(lambda t: t.hour)
```

Distance and Time Visualization

The longer the distance between pickup and dropoff locations, the higher the fare.

Some trips, like to/from an airport are fixed fee.

Fare at night is different from the day time.

Formula to be used for calculating the distance between latitude and longitude

In [38]:
```
def distance(lat1, lon1, lat2, lon2):
  p = 0.017453292519943295 # Pi/180
  a = 0.5 - np.cos((lat2 - lat1) * p)/2 + np.cos(lat1 * p) * np.cos(lat2 * p) * (
  return 0.6213712 * 12742 * np.arcsin(np.sqrt(a))
```

The longer the distance between pickup and dropoff location, higher the fare. Adding new column to dataframe with distance in miles

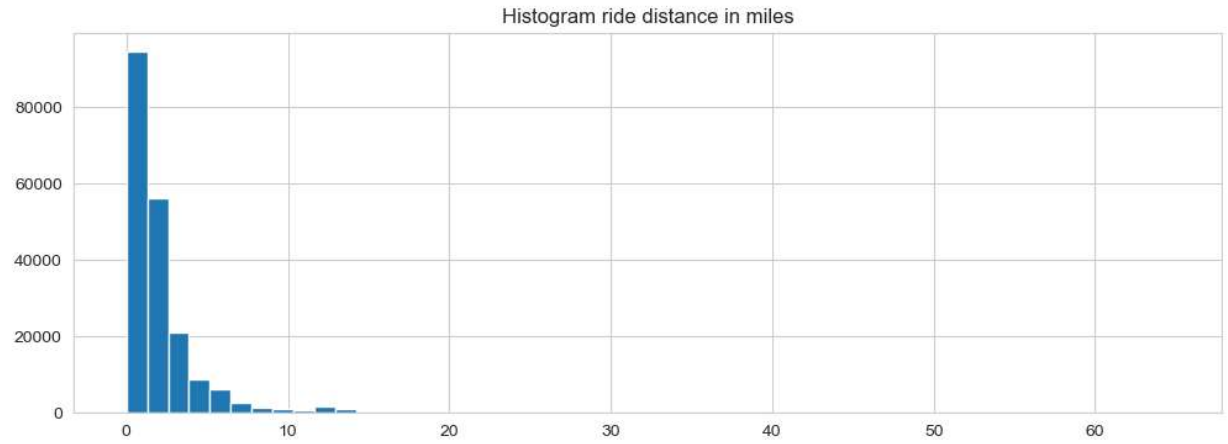
In [39]:
```
df_train["distance_miles"] = distance(df_train["pickup_latitude"], df_train["pick
                                      df_train["dropoff_latitude"], df_train["dro
```

In [40]:
```
df_train.head(2)
```

Out[40]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitu
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.8416
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.9792

In [41]:
```
df_train["distance_miles"].hist(bins=50, figsize=(12,4))
plt.title("Histogram ride distance in miles");
```



In [42]:
```
df_train["distance_miles"].describe()
```

Out[42]:
```
count    195612.000000
mean          2.063410
std           2.352008
min           0.000000
25%           0.780419
50%           1.337858
75%           2.423327
max          64.644331
Name: distance_miles, dtype: float64
```

It seems most rides are just short rides, with a small peak at ~13 miles. This peak could be due to airport drives.

In [43]:
```
df_train.groupby('passenger_count')['distance_miles','fare_amount'].mean()
```

```
C:\Users\shiva\AppData\Local\Temp\ipykernel_16432\1226403062.py:1: FutureWarnin
g: Indexing with multiple keys (implicitly converted to a tuple of keys) will be
deprecated, use a list instead.
  df_train.groupby('passenger_count')['distance_miles','fare_amount'].mean()
```

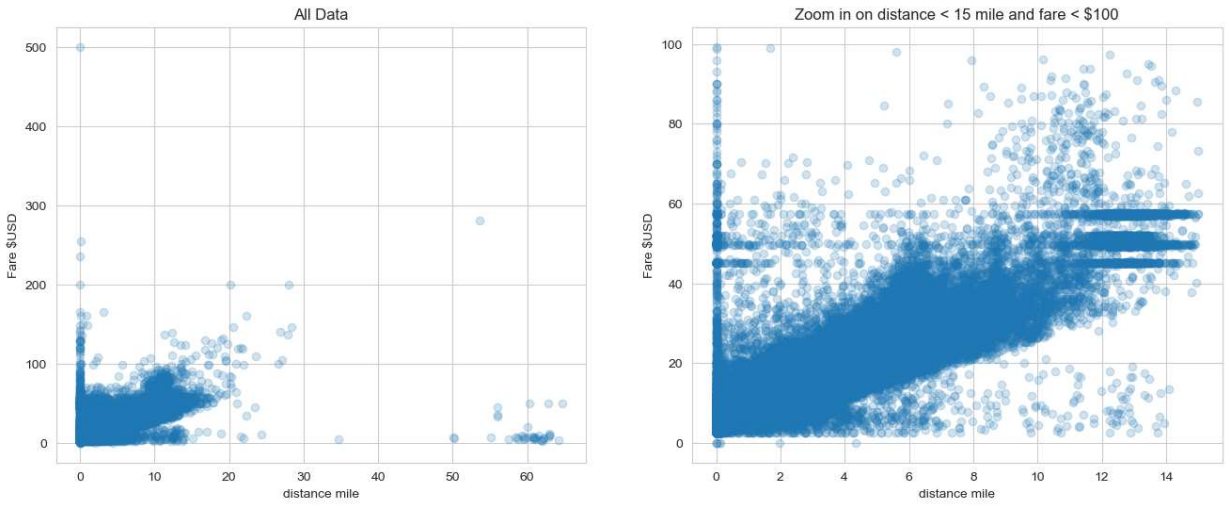
Out[43]:

passenger_count	distance_miles	fare_amount
0	1.836291	9.080659
1	2.035126	11.169553
2	2.174035	11.786048
3	2.070869	11.332867
4	2.126675	11.680824
5	2.066121	11.172870
6	2.168814	12.354238

Scatter Plot distance vs Fare

In [44]:
```
fig, axs = plt.subplots(1, 2, figsize=(16,6))
axs[0].scatter(df_train["distance_miles"], df_train["fare_amount"], alpha=0.2)
axs[0].set_xlabel("distance mile")
axs[0].set_ylabel("Fare $USD")
axs[0].set_title("All Data")

# Zoom-in some part of the data
idx = ((df_train['distance_miles'] < 15) & (df_train["fare_amount"] < 100))
axs[1].scatter(df_train[idx]["distance_miles"], df_train[idx]["fare_amount"], alp
axs[1].set_xlabel("distance mile")
axs[1].set_ylabel("Fare $USD")
axs[1].set_title("Zoom in on distance < 15 mile and fare < $100")
```

Out[44]:
```
Text(0.5, 1.0, 'Zoom in on distance < 15 mile and fare < $100')
```



Few Obervations -

There are trips with zero distance but with a non-zero fare. Could this be trips from and to the same location? Predicting these fares will be difficult as there is likely not sufficient information in the dataset.

There are some trips with >50 miles travel distance but low fare.Perhaps these are discounted trips

The horizontal lines in the right plot might indicate again the fixed fare trips to/from JFK airport.

Overall there seems to be a (linear) relation between distance and fare with an average rate of +/- 100/20 = 5 $USD/mile.

In [45]:
```
# remove datapoints with distance <0.05 miles
idx = (df_train["distance_miles"] >= 0.05)
print('Old size: %d' % len(df_train))
df_train = df_train[idx]
print('New size: %d' % len(df_train))
```

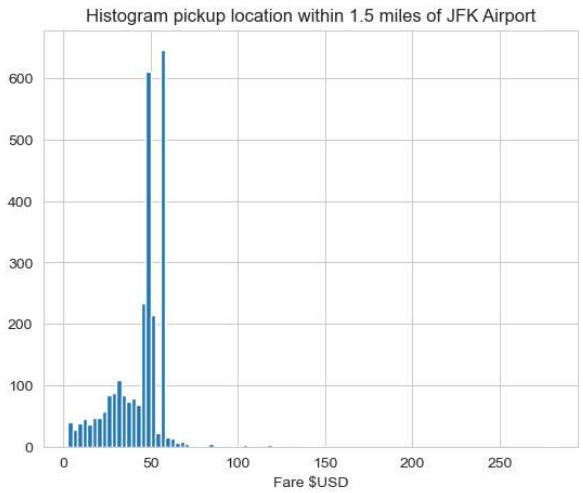
```
Old size: 195612
New size: 192447
```

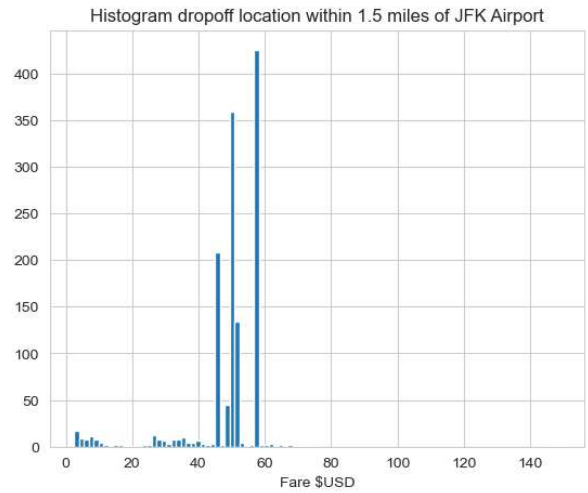
JKF airport coordinates, see https://www.travelmath.com/airport/JFK (https://www.travelmath.com/airport/JFK)

In [46]:
```
jfk = (-73.7822222222,40.6441666667)
nyc = (-74.0063889, 40.7141667)
```

In [47]:
```
def plot_location_fare(loc, name, range=1.5):
    # select all datapoints with dropoff location within range of airport
    fig, axs = plt.subplots(1,2, figsize=(14,5))
    idx = (distance(df_train["pickup_latitude"], df_train["pickup_longitude"], lo
    df_train[idx]["fare_amount"].hist(bins = 100, ax=axs[0])
    axs[0].set_xlabel("Fare $USD")
    axs[0].set_title("Histogram pickup location within {} miles of {}".format(ran

    idx = (distance(df_train["dropoff_latitude"], df_train["dropoff_longitude"],
    df_train[idx]["fare_amount"].hist(bins=100, ax=axs[1])
    axs[1].set_xlabel("Fare $USD")
    axs[1].set_title("Histogram dropoff location within {} miles of {}".format(ra
```

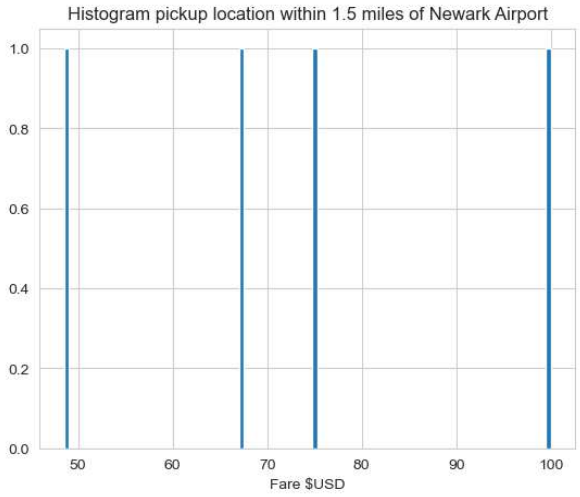
In [48]:
```
plot_location_fare(jfk, 'JFK Airport')
```

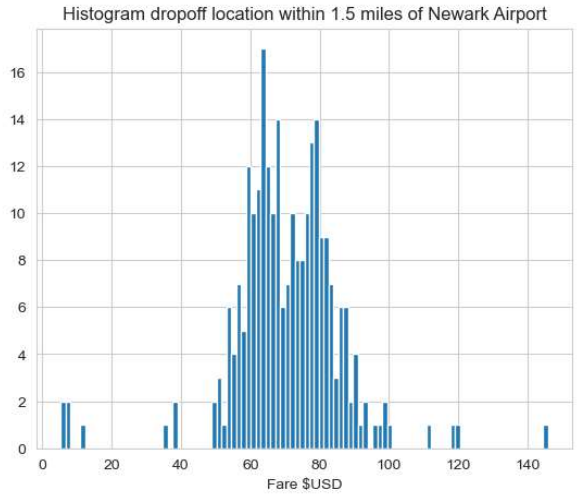




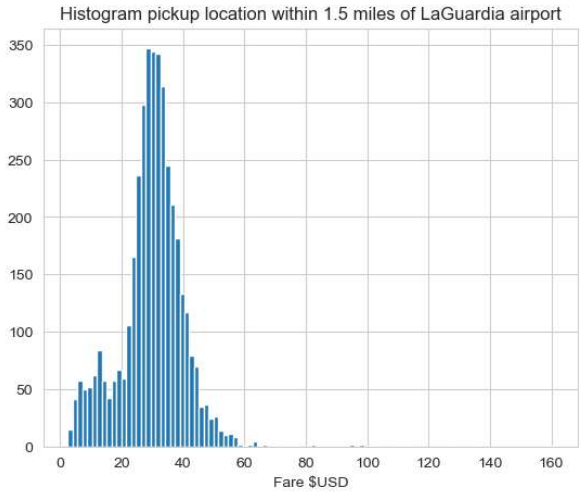
In [49]:
```
#Other Airports
ewr = (-74.175, 40.69) #Newark Liberty International Airport https://www.travelma
lgr = (-73.87, 40.77) #Laguardia Airport, https://www.travelmath.com/airport/LGA
```

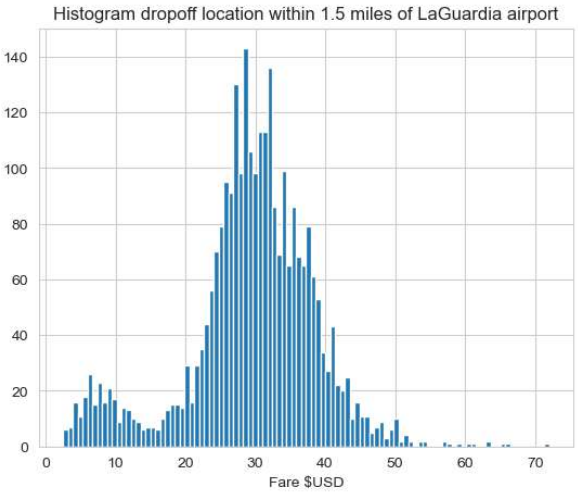
In [50]:
```
plot_location_fare(ewr, 'Newark Airport')
```





In [51]:
```
plot_location_fare(lgr, 'LaGuardia airport')
```





Fare at night is different from the day time

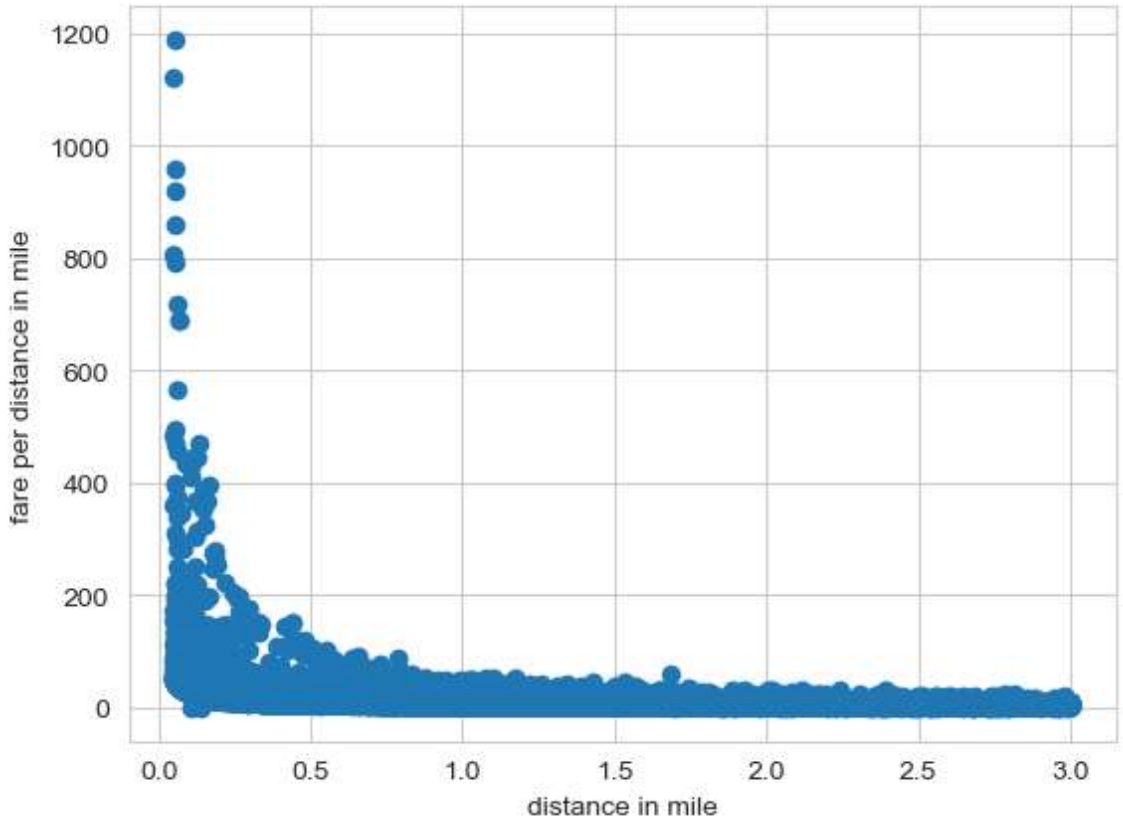
In [52]:
```
df_train["fare_per_mile"] = df_train["fare_amount"] / df_train["distance_miles"]
```

In [53]:
```
df_train["fare_per_mile"].describe()
```

Out[53]:
```
count    192447.000000
mean          7.404693
std          15.918010
min           0.000000
25%           4.771146
50%           6.137379
75%           8.072801
max        3812.571628
Name: fare_per_mile, dtype: float64
```

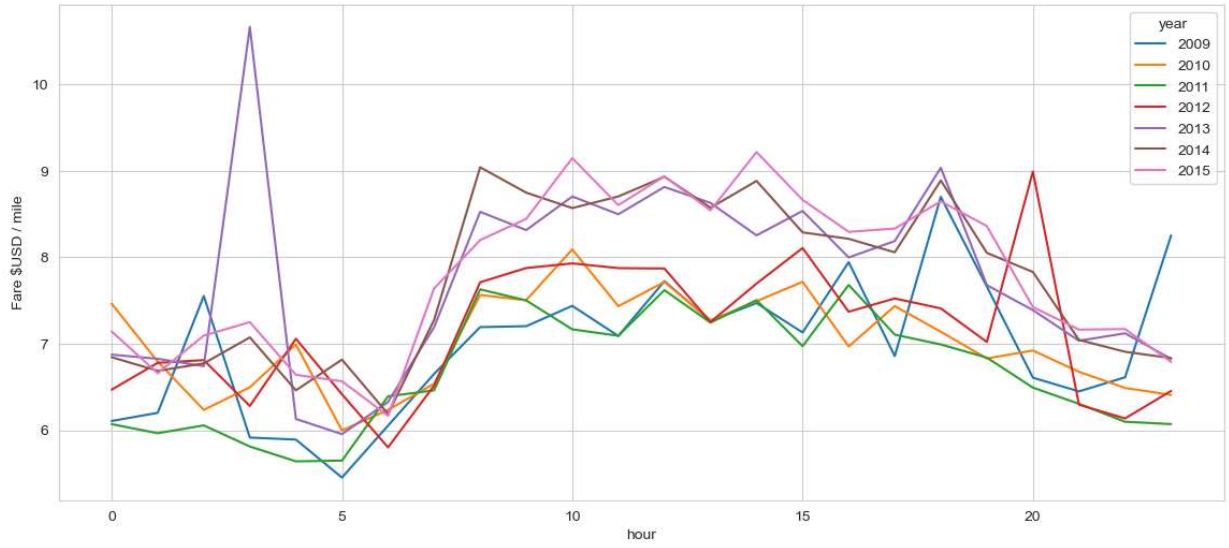
In [54]:
```
idx = (df_train["distance_miles"] < 3) & (df_train["fare_amount"] < 100)
plt.scatter(df_train[idx]["distance_miles"], df_train[idx]["fare_per_mile"])
plt.xlabel("distance in mile")
plt.ylabel("fare per distance in mile")
```

Out[54]:
```
Text(0, 0.5, 'fare per distance in mile')
```



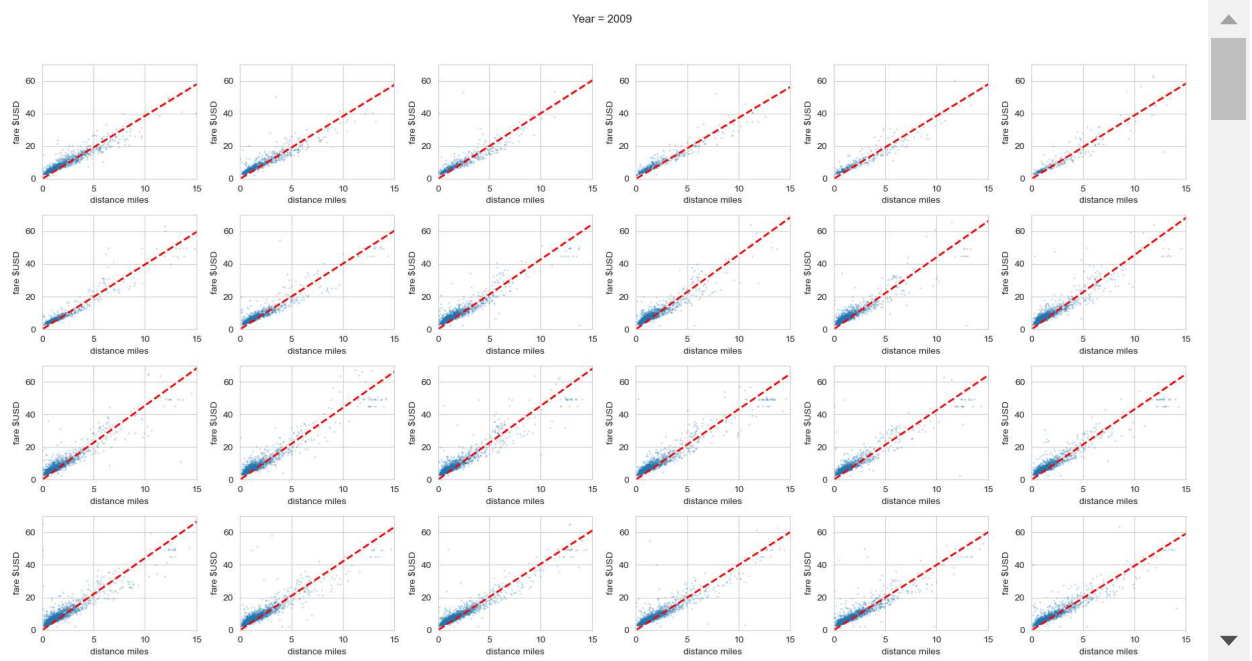
Pivot Table

In [55]:
```
df_train.pivot_table("fare_per_mile", index="hour", columns="year").plot(figsize=
plt.ylabel("Fare $USD / mile");
```



Fare and Time Dependency Per Year

In [56]:
```
from sklearn.linear_model import LinearRegression

# plot all years
for year in df_train["year"].unique():

    # create figure
    fig, axs = plt.subplots(4, 6, figsize=(18, 10))
    axs = axs.ravel()

    # plot for all hours

    for h in range(24):
        idx = (df_train["distance_miles"] < 15) & (df_train["fare_amount"] < 100)
              (df_train["year"] == year)
        axs[h].scatter(df_train[idx]["distance_miles"], df_train[idx]["fare_amoun
        axs[h].set_xlabel('distance miles')
        axs[h].set_ylabel('fare $USD')
        axs[h].set_xlim((0, 15))
        axs[h].set_ylim((0, 70))

        model = LinearRegression(fit_intercept=False)

        X, y = df_train[idx]["distance_miles"].values.reshape(-1,1), df_train[idx
        model.fit(X, y)
        xx = np.linspace(0.1, 25, 100)
        axs[h].plot(xx, model.predict(xx.reshape(-1,1)), '--', c='r', lw=2)

    plt.suptitle("Year = {}".format(year))
    plt.tight_layout(rect=[0, 0, 1, 0.95]);
```



Relevance of direction for calculation of fare amount

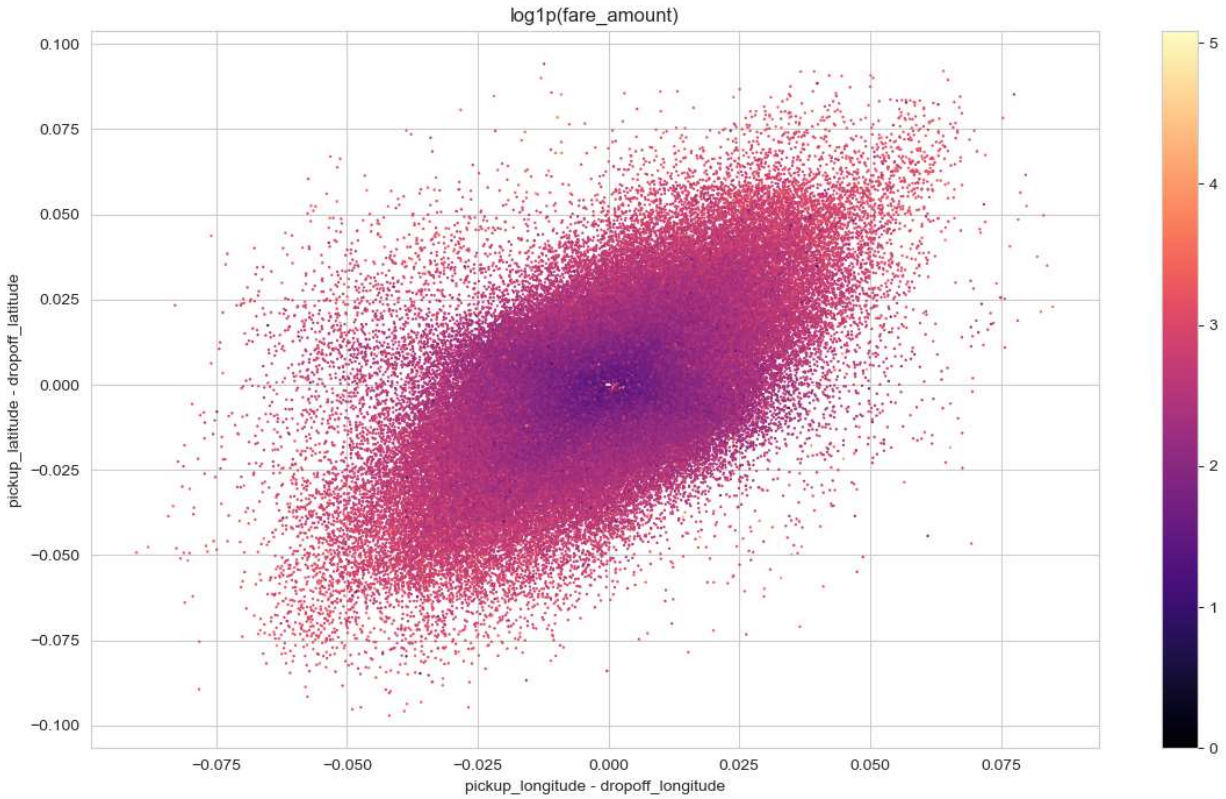
In [57]:
```
df_train["delta_lon"] = df_train["pickup_longitude"] - df_train["dropoff_longitud
df_train["delta_lat"] = df_train["pickup_latitude"] - df_train["dropoff_latitude"
```

Select trips in Manhattan

In [58]:
```
BB_manhattan = (-74.025, -73.925, 40.7, 40.8)
idx_manhattan = select_within_boundingbox(df_train, BB_manhattan)
```

In [59]:
```
plt.figure(figsize=(14,8))
plt.scatter(df_train[idx_manhattan]["delta_lon"], df_train[idx_manhattan]["delta_
            c=np.log1p(df_train[idx_manhattan]["fare_amount"]), cmap="magma")
plt.colorbar()
plt.xlabel('pickup_longitude - dropoff_longitude')
plt.ylabel('pickup_latitude - dropoff_latitude')
plt.title('log1p(fare_amount)')
```

Out[59]: Text(0.5, 1.0, 'log1p(fare_amount)')

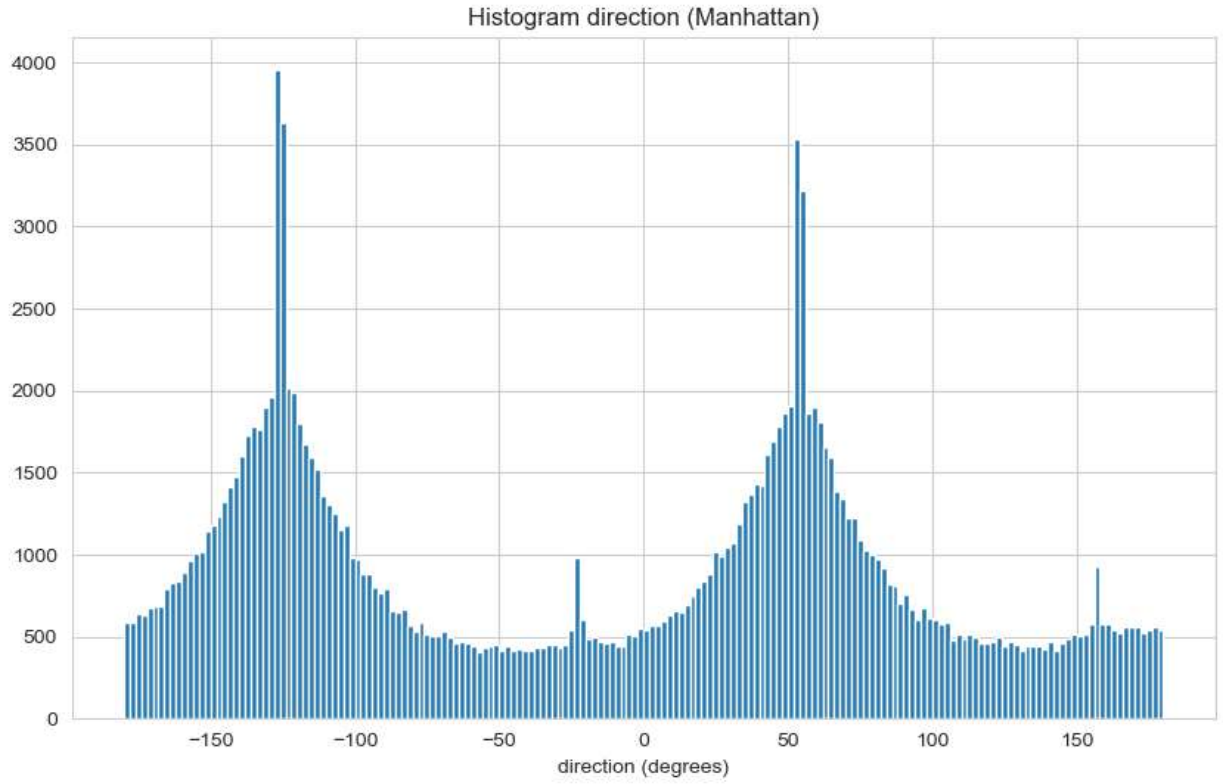


Looks like direction of the trip seems to matter. Direction of a trip, from 180 to -180 degrees. Horizontal axes = 0 degrees

In [60]:
```
def calculate_direction(d_lon, d_lat):
    result = np.zeros(len(d_lon))
    l = np.sqrt(d_lon**2 + d_lat**2)
    result[d_lon>0] = (180/np.pi)*np.arcsin(d_lat[d_lon>0]/l[d_lon>0])
    idx = (d_lon<0) & (d_lat>0)
    result[idx] = 180 - (180/np.pi)*np.arcsin(d_lat[idx]/l[idx])
    idx = (d_lon<0) & (d_lat<0)
    result[idx] = -180 - (180/np.pi)*np.arcsin(d_lat[idx]/l[idx])
    return result
```

In [61]:
```
df_train['direction'] = calculate_direction(df_train.delta_lon, df_train.delta_la
```

In [62]:
```
# plot histogram of directions
plt.figure(figsize=(10,6))
df_train[idx_manhattan].direction.hist(bins=180)
plt.xlabel('direction (degrees)')
plt.title('Histogram direction (Manhattan)')

# plot direction vs average fare amount
fig, ax = plt.subplots(1, 1, figsize=(14,6))
direc = pd.cut(df_train[idx_manhattan]['direction'], np.linspace(-180, 180, 40))

df_train[idx_manhattan].pivot_table('fare_amount', index=[direc], columns='year',

plt.xlabel('direction (degrees)')
plt.xticks(range(36), np.arange(-170, 190, 10))
plt.ylabel('average fare amount $USD');
```



Fare varies with pickup location

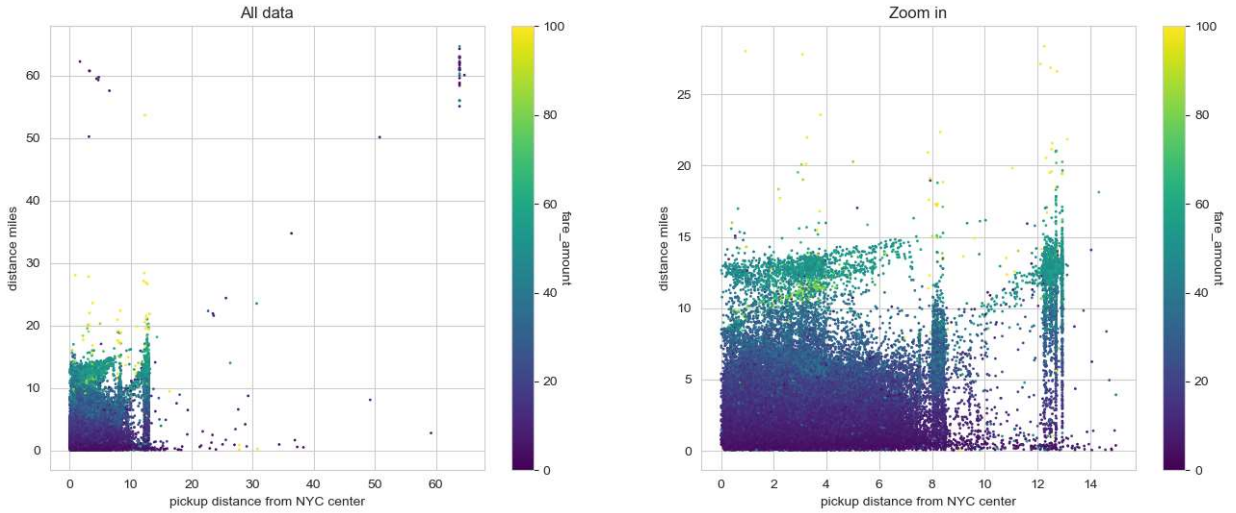
In [63]:
```
# add new column to dataframe with distance in mile
df_train['distance_to_center'] = distance(nyc[1], nyc[0], df_train["pickup_latitu
```

In [64]:
```
df_train.head()
```

Out[64]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longi
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.84
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.97
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270	-73.99
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143	-73.99
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008	-73.95

In [65]:
```
fig, axs = plt.subplots(1, 2, figsize=(16,6))
im = axs[0].scatter(df_train["distance_to_center"], df_train["distance_miles"], c
                    cmap='viridis', alpha=1.0, s=1)

axs[0].set_xlabel('pickup distance from NYC center')
axs[0].set_ylabel('distance miles')
axs[0].set_title('All data')

cbar = fig.colorbar(im, ax=axs[0])
cbar.ax.set_ylabel('fare_amount', rotation=270)


idx = (df_train["distance_to_center"] < 15) & (df_train["distance_miles"] < 35)
im = axs[1].scatter(df_train[idx]["distance_to_center"], df_train[idx]["distance_
                    c=np.clip(df_train[idx]["fare_amount"], 0, 100), cmap='virid
axs[1].set_xlabel('pickup distance from NYC center')
axs[1].set_ylabel('distance miles')
axs[1].set_title('Zoom in')
cbar = fig.colorbar(im, ax=axs[1])
cbar.ax.set_ylabel('fare_amount', rotation=270);
```



There is a lot of 'green' dots, which is about 50 TO 60 fare amount near 13 miles distance of NYC center of distrance of trip. This could be due to trips from/to JFK airport

Baseline Model and Submission

In [66]:
```
# add new column to dataframe with distance in km
df_test['distance_miles'] = distance(df_test["pickup_latitude"], df_test["pickup_
                                     df_test["dropoff_latitude"], df_test["dropof
df_test['distance_to_center'] = distance(nyc[1], nyc[0], df_test["dropoff_latitud
df_test['hour'] = df_test["pickup_datetime"].apply(lambda t: pd.to_datetime(t).ho
df_test['year'] = df_test["pickup_datetime"].apply(lambda t: pd.to_datetime(t).ye
df_test['weekday'] = df_test["pickup_datetime"].apply(lambda t: pd.to_datetime(t)
```

In [67]:
```
df_train.head()
```

Out[67]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longi
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.84
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.97
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270	-73.99
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143	-73.99
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008	-73.95

In [68]:
```
df_test.head()
```

Out[68]:

	key	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pa
0	08:24.0	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	
1	08:24.0	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	
2	53:44.0	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	
3	12:12.0	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	
4	12:12.0	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	

In [69]:
```
# define dataset
# select points 15 miles near NYC center and remove zero passenger datapoints
idx = (df_train["distance_to_center"] < 15) & (df_train["passenger_count"] != 0)

features = ['year', 'hour', 'distance_miles', 'passenger_count','weekday','distan

X = df_train[idx][features].values
y = df_train[idx]['fare_amount'].values
```

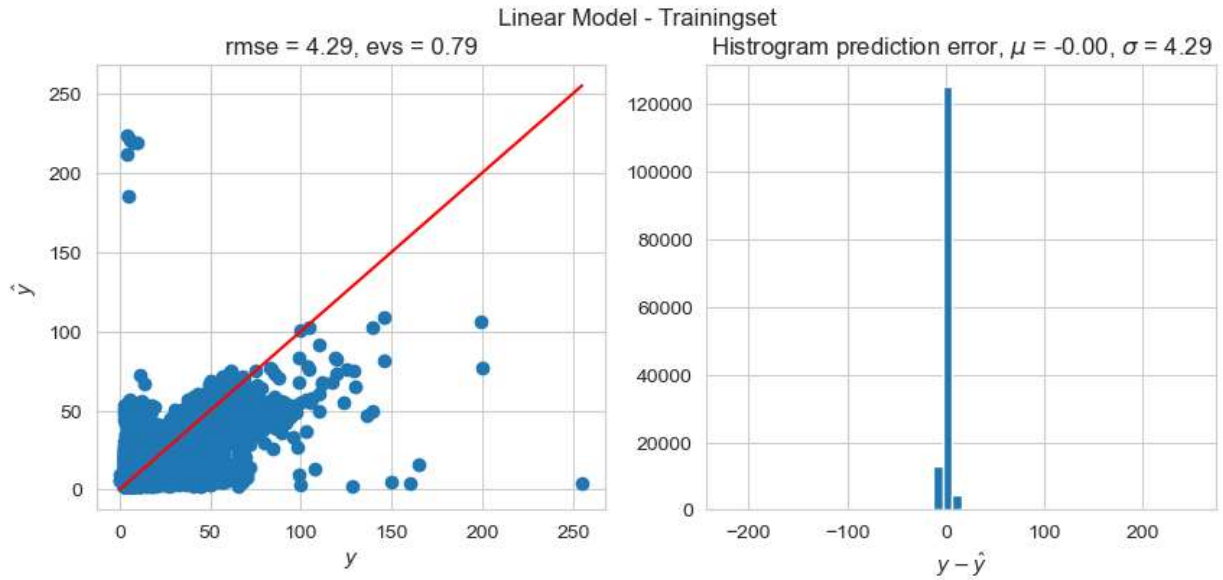
In [70]:
```
print(X.shape, y.shape)
```

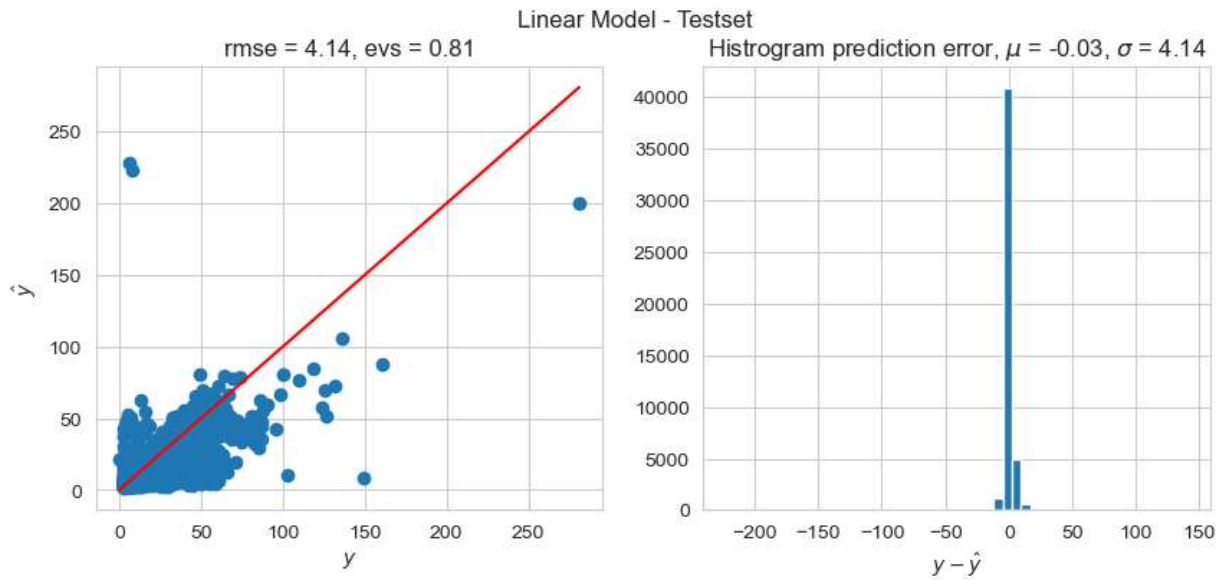
```
(191672, 6) (191672,)
```

In [71]:
```
# define some handy analysis support function
from sklearn.metrics import mean_squared_error, explained_variance_score

def plot_prediction_analysis(y, y_pred, figsize=(10,4), title=''):
    fig, axs = plt.subplots(1, 2, figsize=figsize)
    axs[0].scatter(y, y_pred)
    mn = min(np.min(y), np.min(y_pred))
    mx = max(np.max(y), np.max(y_pred))
    axs[0].plot([mn, mx], [mn, mx], c='red')
    axs[0].set_xlabel('$y$')
    axs[0].set_ylabel('$\hat{y}$')
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    evs = explained_variance_score(y, y_pred)
    axs[0].set_title('rmse = {:.2f}, evs = {:.2f}'.format(rmse, evs))

    axs[1].hist(y-y_pred, bins=50)
    avg = np.mean(y-y_pred)
    std = np.std(y-y_pred)
    axs[1].set_xlabel('$y - \hat{y}$')
    axs[1].set_title('Histrogram prediction error, $\mu$ = {:.2f}, $\sigma$ = {:.

    if title!='':
        fig.suptitle(title)
```

In [72]:
```
# create training and test sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

In [73]:
```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

model_lin = Pipeline((
        ("standard_scaler", StandardScaler()),
        ("lin_reg", LinearRegression()),
    ))

model_lin.fit(X_train, y_train)

y_train_pred = model_lin.predict(X_train)

plot_prediction_analysis(y_train, y_train_pred, title='Linear Model - Trainingset

y_test_pred = model_lin.predict(X_test)
plot_prediction_analysis(y_test, y_test_pred, title='Linear Model - Testset')
```





In [74]:
```
# define dataset
XTEST = df_test[features].values
```

In [82]:
```
y_pred_final = model_lin.predict(XTEST)

submission = pd.DataFrame(
    {
        'key': df_test["key"],
        'fare_amount': y_pred_final
    },
    columns = ['key', 'fare_amount'])
submission.to_csv(r'train(2).csv', index = False)
```

In [83]:
```
submission.head()
```

Out[83]:

	key	fare_amount
0	08:24.0	10.568709
1	08:24.0	10.709545
2	53:44.0	4.424370
3	12:12.0	8.081821
4	12:12.0	15.776577

In [ ]:
```

```

In [80]:
```
sns.heatmap(df_train.corr())
```

```
C:\Users\shiva\AppData\Local\Temp\ipykernel_16432\1376617749.py:1: FutureWarnin
g: The default value of numeric_only in DataFrame.corr is deprecated. In a futur
e version, it will default to False. Select only valid columns or specify the va
lue of numeric_only to silence this warning.
  sns.heatmap(df_train.corr())
```

Out[80]: <Axes: >