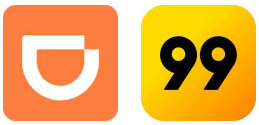


EDA • Exploratory Data Analysis

ISIS SANTOS COSTA • 2021

Exploratory Data Analysis • Resuming: retrieving the DF, final transformed

In [1]:

```
import pandas as pd
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
#!pip install ipython-autotime
%load_ext autotime
```

```
time: 0 ns
```

In [4]:

```
pd.set_option('display.max_columns', None)
```

```
time: 32 ms
```

Data by order

In [6]:

```
with open('df_orders_tF.csv') as f: cols = f.readline().split(',')
del cols[0]
cols[-1] = cols[-1].replace('\n','')
df = pd.read_csv('df_orders_tF.csv', usecols=cols) # reading previously transformed data
df.head(2)
```

Out[6]:

	order_id	passenger_id	pickup_datetime	pickup_month	pickup_week	pickup_weekinmo
0	6433697	5234567816269547	2014-04-11 18:06:45	4	15	
1	6433698	5234567812367422	2014-04-12 00:20:28	4	15	

```
time: 3min 6s
```

In [7]:

```
# bckp, in case there is any issue with our df being improved
with open('df_orders_tF.csv') as f: cols = f.readline().split(',')
del cols[0]
cols[-1] = cols[-1].replace('\n','')
df_original = pd.read_csv('df_orders_tF.csv', usecols=cols) # reading previously transforme
```

```
time: 7min 18s
```

In [9]:

```
print(df.columns == df_original.columns)
```

```
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True]
time: 375 ms
```

Daily data

In [6]:

```
df_daily = pd.read_csv('df_orders_tF_daily.csv') # reading previously transformed data
df_daily.head(2)
```

Out[6]:

	pickup_date	total_orders	total_trips	pickup_month	pickup_week	pickup_weekinmonth	picl
0	2014-03-01	277284	211545	3	9	1	
1	2014-03-02	231264	176557	3	9	1	

```
time: 406 ms
```

Data by passenger

In [7]:

```
df_passenger = pd.read_csv('df_orders_tF_passenger.csv') # reading previously transformed d
df_passenger.head(2)
```

Out[7]:

	passenger_id	total_orders	pickup_month	pickup_week	pickup_weekinmonth	pickup_is
0	5234567812345678	160	3.693750	14.468750	2.718750	
1	5234567812345679	132	3.689394	14.484848	2.727273	

```
time: 33 s
```

Exploratory Data Analysis • Organizing features into classes

👀 *Let's take a look at the features again to choose the relations to be visually inspected.*

Features List • Data by order

In [8]:

```
list(df.columns)
```

Out[8]:

```
['order_id',
 'passenger_id',
 'pickup_datetime',
 'pickup_month',
 'pickup_week',
 'pickup_weekinmonth',
 'pickup_date',
 'pickup_dayofweek',
 'pickup_isweekend',
 'pickup_hour',
 'pickup_isrush',
 'pickup_latitude',
 'pickup_longitude',
 'dropoff_latitude',
 'dropoff_longitude',
 'dropoff_datetime',
 'duration',
 'duration_minutes',
 'duration_type',
 'trip_distance',
 'total_amount',
 'fare_amount',
 'tip_amount',
 'rate_type',
 'payment_type',
 'pickup_borough',
 'dropoff_borough']
```

```
time: 79 ms
```

Features List • Daily data

In [9]:

```
list(df_daily.columns)
```

Out[9]:

```
['pickup_date',
 'total_orders',
 'total_trips',
 'pickup_month',
 'pickup_week',
 'pickup_weekinmonth',
 'pickup_dayofweek',
 'pickup_isweekend',
 'pickup_hour',
 'pickup_isrush',
 'pickup_latitude',
 'pickup_longitude',
 'dropoff_latitude',
 'dropoff_longitude',
 'duration_minutes',
 'duration_type',
 'trip_distance',
 'total_amount',
 'fare_amount',
 'tip_amount',
 'Standard',
 'JFK',
 'Newark',
 'Nassau / Westchester',
 'Negotiated',
 'Group ride',
 'CRD',
 'CSH',
 'UNK',
 'NOC',
 'DIS',
 'pickup ‖ Manhattan • Lower Manhattan',
 'pickup ‖ Manhattan • Midtown Manhattan',
 'pickup ‖ Manhattan • Upper Manhattan',
 'pickup ‖ Queens',
 'pickup ‖ Brooklyn',
 'pickup ‖ Manhattan • Harlem',
 'pickup ‖ Manhattan • Washington Heights',
 'pickup ‖ Bronx',
 'pickup ‖ Staten Island',
 'dropoff ‖ Manhattan • Lower Manhattan',
 'dropoff ‖ Manhattan • Midtown Manhattan',
 'dropoff ‖ Manhattan • Upper Manhattan',
 'dropoff ‖ Queens',
 'dropoff ‖ Brooklyn',
 'dropoff ‖ Manhattan • Harlem',
 'dropoff ‖ Manhattan • Washington Heights',
 'dropoff ‖ Bronx',
 'dropoff ‖ Staten Island']
```

time: 15 ms

Features List • Data by passenger

In [10]:

```
list(df_passenger.columns)
```

Out[10]:

```
['passenger_id',
 'total_orders',
 'pickup_month',
 'pickup_week',
 'pickup_weekinmonth',
 'pickup_isweekend',
 'pickup_hour',
 'pickup_isrush',
 'pickup_latitude',
 'pickup_longitude',
 'dropoff_latitude',
 'dropoff_longitude',
 'duration_minutes',
 'trip_distance',
 'total_amount',
 'fare_amount',
 'tip_amount']
```

```
time: 16 ms
```

Features classes • All orders

Input						Target
Client account	Time	Duration	Distance	Location	Payment type	Happiness Revenue
passenger_id*	**pickup_isweekend**	**duration_type**	**trip_distance**	**pickup_borough**	**payment_type**	**tip_amoun (Happines**
	pickup_isrush	**duration_minutes**		**dropoff_borough**		**total_amou (Revenue)**
	pickup_month	duration		**rate_type****		fare_amour (Revenue)
	pickup_weekinmonth					
	pickup_dayofweek					
	pickup_hour					
	pickup_datetime					
	pickup_week					
	pickup_date					
	dropoff_datetime					

* **bold features are representative of the whole class**

** *rate_type* classified considering it is mostly location-related, only exception being the 'group' ride

Features classes • Daily data

Input						Target
Client transactions	Time	Duration	Distance	Location	Payment type	Happiness Revenue
total_orders*	**pickup_date**	**duration_type**	**trip_distance**	**pickup_borough** (specified)	**payment_type** (specified)	**tip_amount (Happiness**
total_trips	**pickup_isweekend**	**duration_minutes**		**dropoff_borough** (specified)		**total_amou (Revenue)**
	pickup_isrush			**rate_type**** (specified)		fare_amoun (Revenue)
	pickup_month			pickup_latitude		
	pickup_weekinmonth			pickup_longitude		
	pickup_dayofweek			dropoff_latitude		
	pickup_hour			dropoff_longitude		
	pickup_datetime					
	pickup_week					
	dropoff_datetime					

* **bold features are representative of the whole class**
** *rate_type* classified considering it is mostly location-related, only exception being the 'group' ride

Features classes • Data by passenger

Input					Target
Client transactions	Time	Duration	Distance	Location	Happiness & Revenue
total_orders*	**pickup_isweekend**	**duration_type**	**trip_distance**	**pickup_latitude**	**tip_amount (Happiness)**
	pickup_isrush	**duration_minutes**		**pickup_longitude**	**total_amount (Revenue)**
	pickup_month			**dropoff_latitude**	fare_amount (Revenue)
	pickup_weekinmonth			**dropoff_longitude**	
	pickup_hour				
	pickup_week				

* **bold features are representative of the whole class**

Exploratory Data Analysis • Importing matplotlib

In [5]:

```
%matplotlib inline

import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.style.use('ggplot')
```

```
time: 6.16 s
```

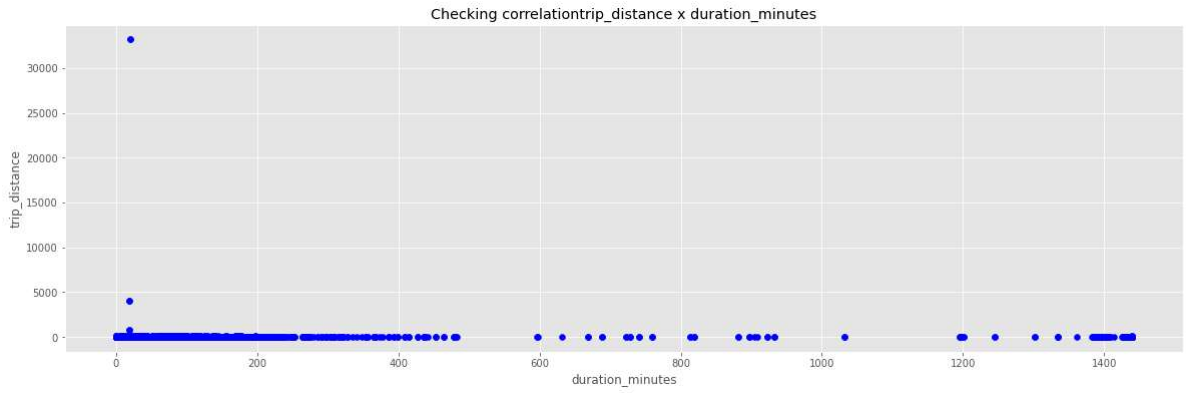
Exploratory Data Analysis • Raw data (by order) ǁ Correlations

🔎 *By searching for correlations, we can understand pairwise interactions between variables and also check for possible reductions towards* ***data efficiency****.*

Predictor x Predictor: Trip duration *vs.* distance

In [13]:

```
# Checking correlation trip duration x distance
x, y = df['duration_minutes'], df['trip_distance']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation'+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```




```
time: 26.6 s
```

Those two points below 30 minutes, close to 4,000 miles and 34,000 miles, respectively, correspond to around ***8k miles/hour*** *and* ***~70k miles/hour****.*

Even for some extravagant scientific experiment of maybe Tesla visionaires this seem a little too much: safe to drop. 🧹

In [10]:

```
indexNames = df[ (df['trip_distance'] > 2000) ].index
df.drop(indexNames, inplace=True)
max(df['trip_distance'])
```

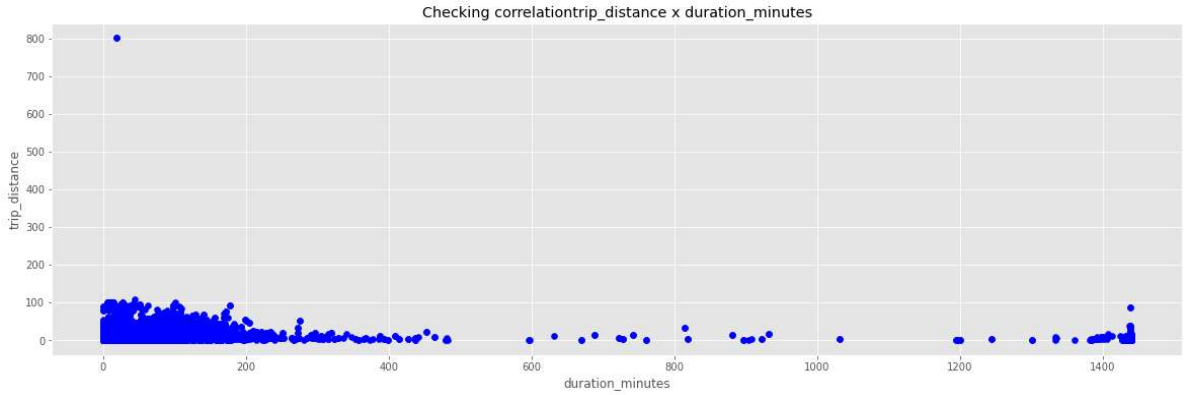
Out[10]:

```
800.5
```

time: 9min 34s

In [15]:

```
# Checking correlation trip duration x distance
x, y = df['duration_minutes'], df['trip_distance']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation'+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```



time: 1min 3s

1,600 miles/hour still not credible: let's drop further.

In [11]:

```
indexNames = df[ (df['trip_distance'] > 800) ].index
df.drop(indexNames, inplace=True)
max(df['trip_distance'])
```

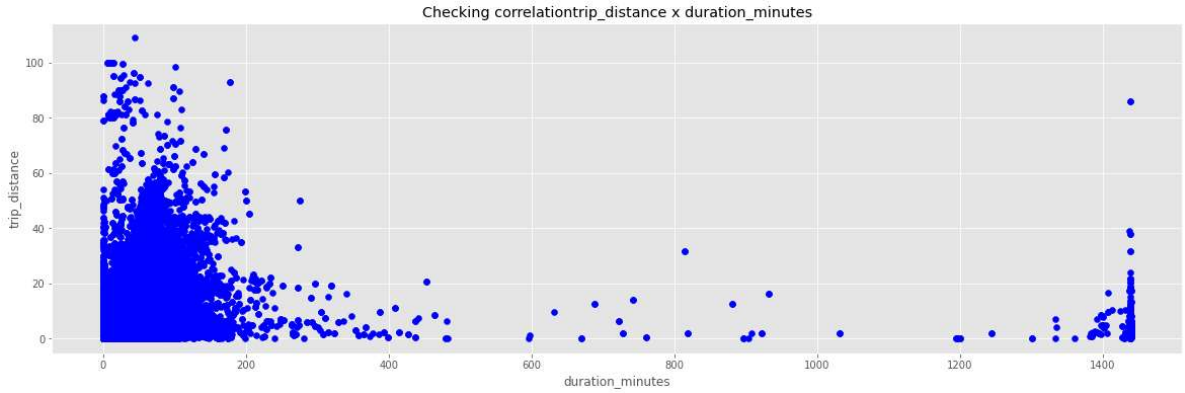
Out[11]:

```
108.9
```

time: 12min 45s

In [17]:

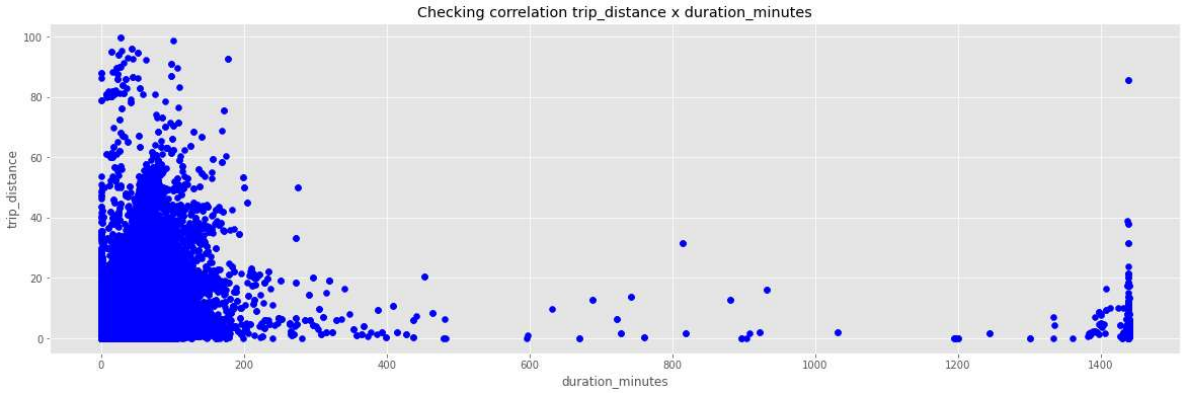
```
# Checking correlation trip duration x distance
x, y = df['duration_minutes'], df['trip_distance']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation'+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```



time: 35.1 s

In [178]:

```
# Checking correlation trip duration x distance: filtered by distance < threshold
threshold = 100
x, y = df.loc[df['trip_distance'] < threshold, 'duration_minutes'], df.loc[df['trip_distanc

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```

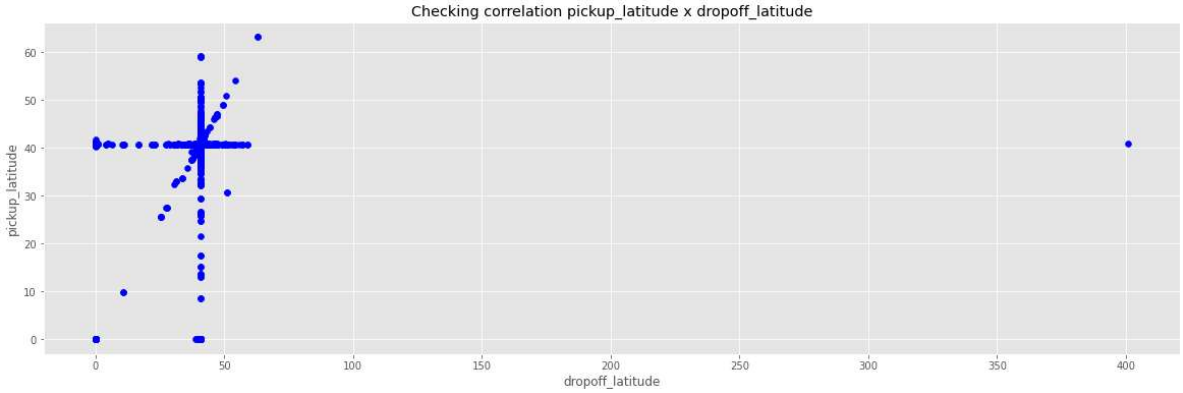


time: 1min 12s

Predictor x Predictor: Dropoff latitude *vs.* Pickup latitude

In [19]:

```
# Checking correlation Dropoff Latitude x Pickup Latitude
x, y = df['dropoff_latitude'], df['pickup_latitude']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```



time: 27.4 s

🤔 *Linear interactions display a curious pattern.*

🛠 *The point showing around 400 for dropoff latitude value is wrong, let's fix 'dropoff_latitude' to the valid*

range: -90° to 90°.

In [180]:

```
df['dropoff_latitude'].describe()
```

Out[180]:

```
count    1.145622e+07
mean     3.997215e+01
std      5.581361e+00
min      0.000000e+00
25%      4.073346e+01
50%      4.075306e+01
75%      4.076830e+01
max      4.009167e+02
Name: dropoff_latitude, dtype: float64

time: 1.23 s
```

In [13]:

```
df['dropoff_latitude'] = df.loc[df['dropoff_latitude'] <= 90, 'dropoff_latitude']
df['dropoff_latitude'].describe()
```

Out[13]:

```
count    1.145622e+07
mean     3.997212e+01
std      5.580342e+00
min      0.000000e+00
25%      4.073346e+01
50%      4.075306e+01
75%      4.076830e+01
max      6.312818e+01
Name: dropoff_latitude, dtype: float64

time: 11.6 s
```

In [14]:

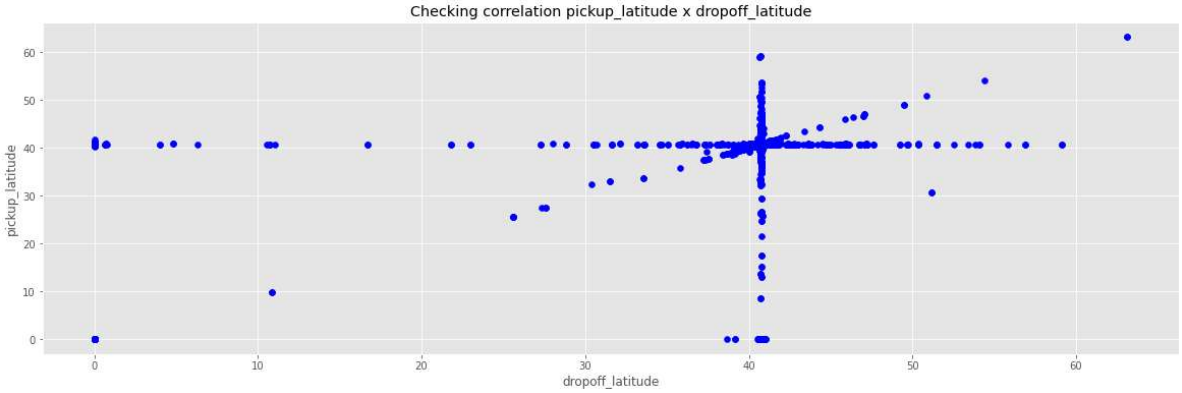
```
df.count()
```

Out[14]:

```
order_id              14999994
passenger_id          14999994
pickup_datetime       14999994
pickup_month          14999994
pickup_week           14999994
pickup_weekinmonth    14999994
pickup_date           14999994
pickup_dayofweek      14999994
pickup_isweekend      14999994
pickup_hour           14999994
pickup_isrush         14999994
pickup_latitude       14999994
pickup_longitude      14999994
dropoff_latitude      11456220
dropoff_longitude     11456221
dropoff_datetime      11456221
duration              11456221
duration_minutes      11456221
duration_type         11456221
trip_distance         11456221
total_amount          11456221
fare_amount           11456221
tip_amount            11456221
rate_type             11456221
payment_type          14999994
pickup_borough        14999994
dropoff_borough       11456221
dtype: int64

time: 10.3 s
```

In [183]:

```
# Checking correlation Dropoff latitude x Pickup latitude
x, y = df['dropoff_latitude'], df['pickup_latitude']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```




```
time: 35.5 s
```

🛠 *Although any point in the range -90° to 90° would be valid, many of the points are not credible,* ***reaching outside the US and even crossing waters****.*

So, let's further ***clean data to keep latitudes within the '48 contiguous states continuously above water'****. By this criterion,*
the southernmost point is **24.520833** (https://en.wikipedia.org/wiki/List_of_extreme_points_of_the_United_States).
the northernmost point is **49.384472** (https://en.wikipedia.org/wiki/List_of_extreme_points_of_the_United_States).
(Degrees Decimal Minutes (DDM) converted to Decimal Degrees (DD) here (https://www.pgc.umn.edu/apps/convert/)).

In [15]:

```
df['pickup_latitude'] = df.loc[(df['pickup_latitude'] >= 24.520833) & (df['pickup_latitude'
df['pickup_latitude'].describe()
```

Out[15]:

```
count    1.472689e+07
mean     4.075073e+01
std      3.724275e-02
min      2.479546e+01
25%      4.073633e+01
50%      4.075327e+01
75%      4.076738e+01
max      4.891002e+01
Name: pickup_latitude, dtype: float64

time: 4.62 s
```

In [16]:

```
df['dropoff_latitude'] = df.loc[(df['dropoff_latitude'] >= 24.520833) & (df['dropoff_latitu
df['dropoff_latitude'].describe()
```

Out[16]:

```
count    1.123719e+07
mean     4.075110e+01
std      3.957435e-02
min      2.557423e+01
25%      4.073494e+01
50%      4.075377e+01
75%      4.076858e+01
max      4.922112e+01
Name: dropoff_latitude, dtype: float64


time: 3.47 s
```

In [17]:

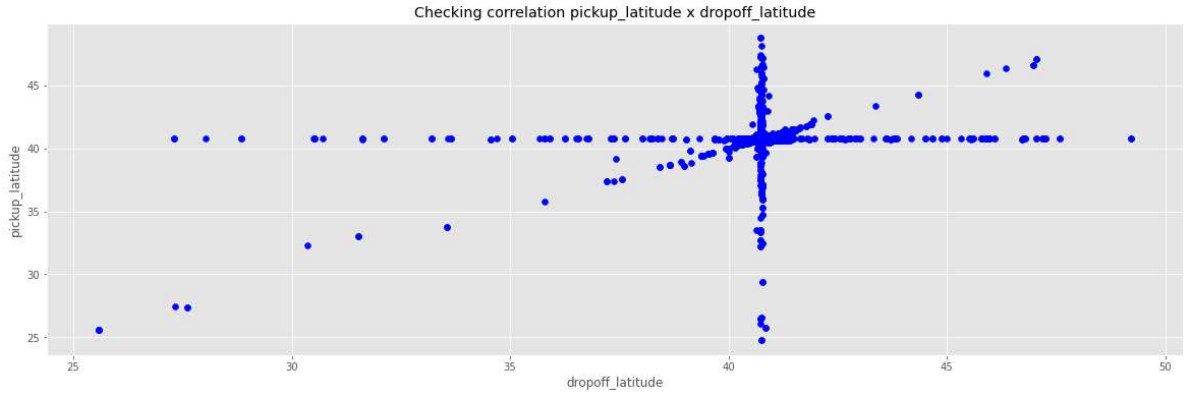
```
df.count()
```

Out[17]:

```
order_id              14999994
passenger_id          14999994
pickup_datetime       14999994
pickup_month          14999994
pickup_week           14999994
pickup_weekinmonth    14999994
pickup_date           14999994
pickup_dayofweek      14999994
pickup_isweekend      14999994
pickup_hour           14999994
pickup_isrush         14999994
pickup_latitude       14726894
pickup_longitude      14999994
dropoff_latitude      11237187
dropoff_longitude     11456221
dropoff_datetime      11456221
duration              11456221
duration_minutes      11456221
duration_type         11456221
trip_distance         11456221
total_amount          11456221
fare_amount           11456221
tip_amount            11456221
rate_type             11456221
payment_type          14999994
pickup_borough        14999994
dropoff_borough       11456221
dtype: int64

time: 8.59 s
```

In [195]:

```
# Checking correlation Dropoff latitude x Pickup latitude
x, y = df['dropoff_latitude'], df['pickup_latitude']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```



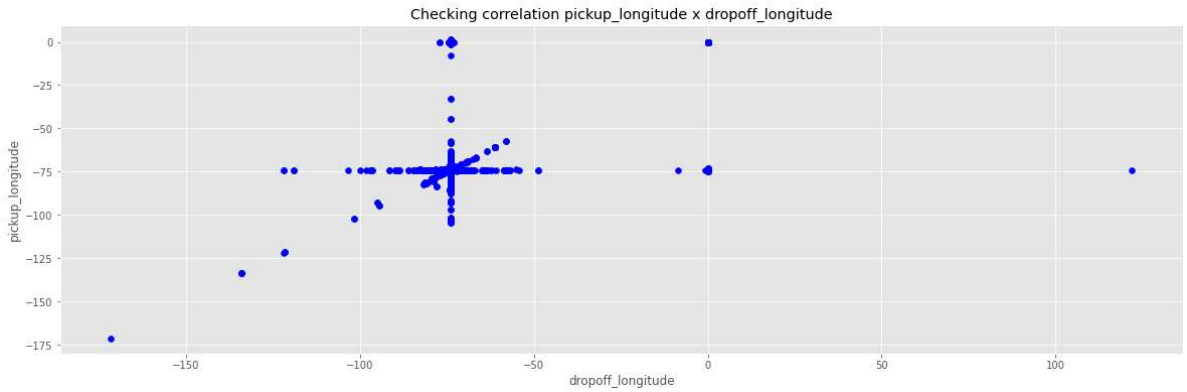

```
time: 33.6 s
```

Nice. ☺

Predictor x Predictor: Dropoff longitude *vs.* Pickup longitude

In [20]:

```
# Checking correlation Dropoff longitude x Pickup longitude
x, y = df['dropoff_longitude'], df['pickup_longitude']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```




```
time: 26.5 s
```

Same as for latitudes, as expected. 🤔

🛠 *The easternmost dropoff longitude at +120° (positive!) would be a destination in China.*
*It would possibly be -120°, but - **as we have plenty of data** - let's simply drop it.*

🛠 *As for latitudes, although any point in the range -180° to 180° would be valid, let's **clean data to keep longitudes within a credible range**:*

Easternmost point on the U.S. mainland: **-66.949778** (https://en.wikipedia.org/wiki/List_of_extreme_points_of_the_United_States).

Westernmost point on the U.S. mainland (contiguous): **-124.733056** (https://en.wikipedia.org/wiki/List_of_extreme_points_of_the_United_States).

(Degrees Decimal Minutes (DDM) converted to Decimal Degrees (DD) here (https://www.pgc.umn.edu/apps/convert/)).

In [10]:

```
df['pickup_longitude'].describe()
```

Out[10]:

```
count    1.500000e+07
mean    -7.262919e+01
std      9.889962e+00
min     -1.714464e+02
25%     -7.399212e+01
50%     -7.398188e+01
75%     -7.396731e+01
max      1.213851e+00
Name: pickup_longitude, dtype: float64
```

```
time: 2.42 s
```

In [18]:

```
df['pickup_longitude'] = df.loc[(df['pickup_longitude'] >= -124.733056) & (df['pickup_longi
df['pickup_longitude'].describe()
```

Out[18]:

```
count    1.472689e+07
mean    -7.397584e+01
std      5.922525e-02
min     -1.219261e+02
25%     -7.399232e+01
50%     -7.398213e+01
75%     -7.396841e+01
max     -6.713012e+01
Name: pickup_longitude, dtype: float64

time: 4.06 s
```

In [12]:

```
df['dropoff_longitude'].describe()
```

Out[12]:

```
count    1.145623e+07
mean    -7.256022e+01
std      1.012975e+01
min     -1.714464e+02
25%     -7.399138e+01
50%     -7.397991e+01
75%     -7.396281e+01
max      1.218270e+02
Name: dropoff_longitude, dtype: float64

time: 921 ms
```

In [19]:

```
df['dropoff_longitude'] = df.loc[(df['dropoff_longitude'] >= -124.733056) & (df['dropoff_lo
df['dropoff_longitude'].describe()
```

Out[19]:

```
count    1.123718e+07
mean    -7.397429e+01
std      6.203673e-02
min     -1.219261e+02
25%     -7.399156e+01
50%     -7.398036e+01
75%     -7.396428e+01
max     -6.743661e+01
Name: dropoff_longitude, dtype: float64

time: 3.42 s
```

In [20]:

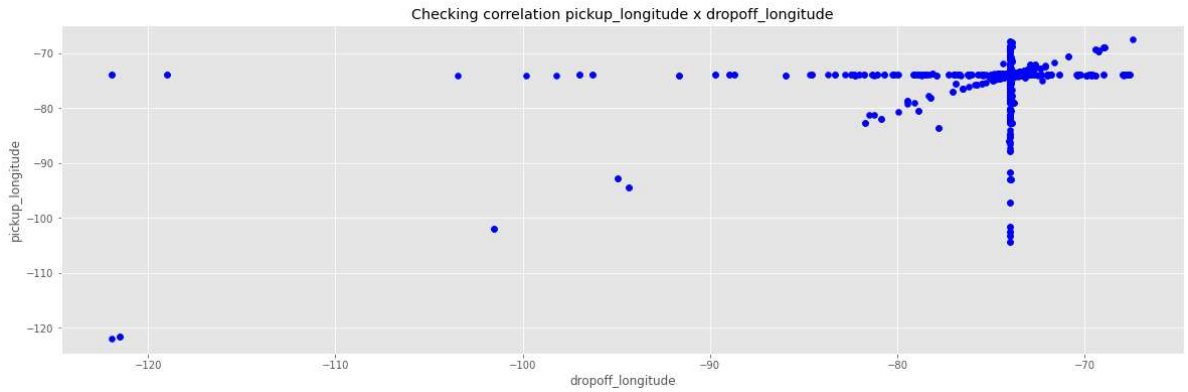
```
df.count()
```

Out[20]:

```
order_id              14999994
passenger_id          14999994
pickup_datetime       14999994
pickup_month          14999994
pickup_week           14999994
pickup_weekinmonth    14999994
pickup_date           14999994
pickup_dayofweek      14999994
pickup_isweekend      14999994
pickup_hour           14999994
pickup_isrush         14999994
pickup_latitude       14726894
pickup_longitude      14726888
dropoff_latitude      11237187
dropoff_longitude     11237178
dropoff_datetime      11456221
duration              11456221
duration_minutes      11456221
duration_type         11456221
trip_distance         11456221
total_amount          11456221
fare_amount           11456221
tip_amount            11456221
rate_type             11456221
payment_type          14999994
pickup_borough        14999994
dropoff_borough       11456221
dtype: int64

time: 9.2 s
```

In [17]:

```
# Checking correlation Dropoff longitude x Pickup longitude
x, y = df['dropoff_longitude'], df['pickup_longitude']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```



time: 26.4 s

Now we see a feasible geographic coverage. ☺

Predictor x Predictor: Dropoff datetime *vs.* Pickup datetime

*When setting variables for feature engineering, we previously decided that - in terms of time - **just** looking at the **pickup** part would be **enough**.*

Now let's have an idea (👀) of what is being left out:

In [21]:

```
df['dropoff_datetime'] = pd.to_datetime(df['dropoff_datetime'])
df['dropoff_datetime'].head(3)
```

Out[21]:

```
0   2014-04-11 18:18:19
1   2014-04-12 00:47:19
2   2014-04-11 23:59:41
Name: dropoff_datetime, dtype: datetime64[ns]
```

time: 14 s

In [22]:

```
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
df['pickup_datetime'].head(3)
```

Out[22]:

```
0   2014-04-11 18:06:45
1   2014-04-12 00:20:28
2   2014-04-11 23:51:28
Name: pickup_datetime, dtype: datetime64[ns]

time: 15.6 s
```

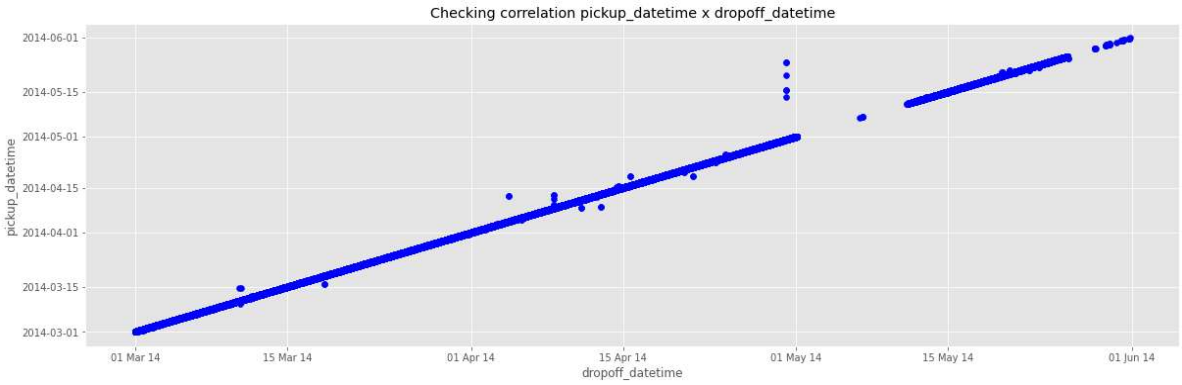
In [23]:

```
print(df['dropoff_datetime'].dtype)
print(df['pickup_datetime'].dtype)
```

```
datetime64[ns]
datetime64[ns]
time: 687 ms
```

In [24]:

```
# Checking correlation Dropoff datetime x Pickup datetime
x, y = df['dropoff_datetime'], df['pickup_datetime']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)

import matplotlib.dates as mdates
dtFmt = mdates.DateFormatter('%d %b %y')
plt.gca().xaxis.set_major_formatter(dtFmt)

plt.show()
```




```
time: 28.9 s
```

💡 ***Strongly linearly correlated, so taking only the pickup_datetime (with derivates) into account seems pretty fine.***

Note: we observe a long period without any trip, them 'a pair of rebels'(?) in what could be interpreted as a brief quarantine for today standards. Yet, there was a concern about Ebola in 2014. For the purpose of integrating present as part of a long-time study, just normalizing *the data would be fine.*

Predictor x Predictor: Speed *vs.* Pickup hour

Typical speeds impact planning operations, so let's add a new feature, 'avg_mph'*, a check its correlation to the hour of the day.*

In [22]:

```
df['mph'] = df['trip_distance'] / df['duration_minutes'] * 60
```

```
time: 7.95 s
```

In [23]:

```
df['mph'].head()
```

Out[23]:

```
0     8.299712
1    12.290503
2    11.683570
3     7.986348
4    10.037679
Name: mph, dtype: float64
```

```
time: 47 ms
```

In [24]:

```
df['mph'].describe()
```

Out[24]:

```
count    1.143461e+07
mean              inf
std               NaN
min      0.000000e+00
25%      8.325000e+00
50%      1.129707e+01
75%      1.530000e+01
max               inf
Name: mph, dtype: float64
```

```
time: 1.83 s
```

Void trips, indicating a ***duration of 0****, are causing erroneous mph values. Let's* ***fix*** *them.*

In [26]:

```
df.loc[df['duration_minutes'] == 0, 'mph'] = 0
```

```
time: 22.1 s
```

In [27]:

```
df['mph'].head()
```

Out[27]:

```
0     8.299712
1    12.290503
2    11.683570
3     7.986348
4    10.037679
Name: mph, dtype: float64

time: 5.52 s
```

In [28]:

```
df['mph'].describe()
```

Out[28]:

```
count    1.145622e+07
mean     1.367559e+01
std      1.304318e+02
min      0.000000e+00
25%      8.300000e+00
50%      1.127610e+01
75%      1.526250e+01
max      1.263600e+05
Name: mph, dtype: float64

time: 5.56 s
```

Much better. ☺

🛠 *But still, there are **unrealistic mph values** in our dataset:*

- *Trips @ **ZERO mph** (calculated from ZERO distance)*
- *Trips reaching **123,600 mph***

*Let's **drop** them. The **high limit speed** will be 45% of that of the 2010 record breaking* Bugatti Veyron 16.4 Super Sport*: 45% of **267.856 mph** (https://en.m.wikipedia.org/wiki/Production_car_speed_record)*

In [29]:

```
df['mph'] = df.loc[(df['mph'] != 0) & (df['mph'] <= (0.45 * 267.856)), 'mph']
df['mph'].describe()
```

Out[29]:

```
count    1.138125e+07
mean     1.269757e+01
std      6.549215e+00
min      1.007078e-02
25%      8.357143e+00
50%      1.130890e+01
75%      1.530000e+01
max      1.200000e+02
Name: mph, dtype: float64

time: 6.03 s
```

Great! The case for lowest mph should still be further checked, the highest value is now an excutable one. ☺

In [30]:

```
df.count()
```

Out[30]:

```
order_id              14999994
passenger_id          14999994
pickup_datetime       14999994
pickup_month          14999994
pickup_week           14999994
pickup_weekinmonth    14999994
pickup_date           14999994
pickup_dayofweek      14999994
pickup_isweekend      14999994
pickup_hour           14999994
pickup_isrush         14999994
pickup_latitude       14726894
pickup_longitude      14726888
dropoff_latitude      11237187
dropoff_longitude     11237178
dropoff_datetime      11456221
duration              11456221
duration_minutes      11456221
duration_type         11456221
trip_distance         11456221
total_amount          11456221
fare_amount           11456221
tip_amount            11456221
rate_type             11456221
payment_type          14999994
pickup_borough        14999994
dropoff_borough       11456221
mph                   11381250
dtype: int64

time: 13.1 s
```

In [31]:

```
df_avg_mph = df[['pickup_hour','mph']].groupby(['pickup_hour']).mean()
df_avg_mph
```

Out[31]:

pickup_hour	mph
0	15.580295
1	16.114417
2	16.617019
3	17.227740
4	19.455687
5	21.532642
6	17.747107
7	13.774582
8	11.319771
9	10.978287
10	11.170708
11	10.920116
12	10.760997
13	10.950791
14	10.816172
15	10.732087
16	11.428234
17	11.194912
18	11.111478
19	11.891611
20	12.947226
21	13.780544
22	14.108880
23	14.753499

time: 4.69 s

In [32]:

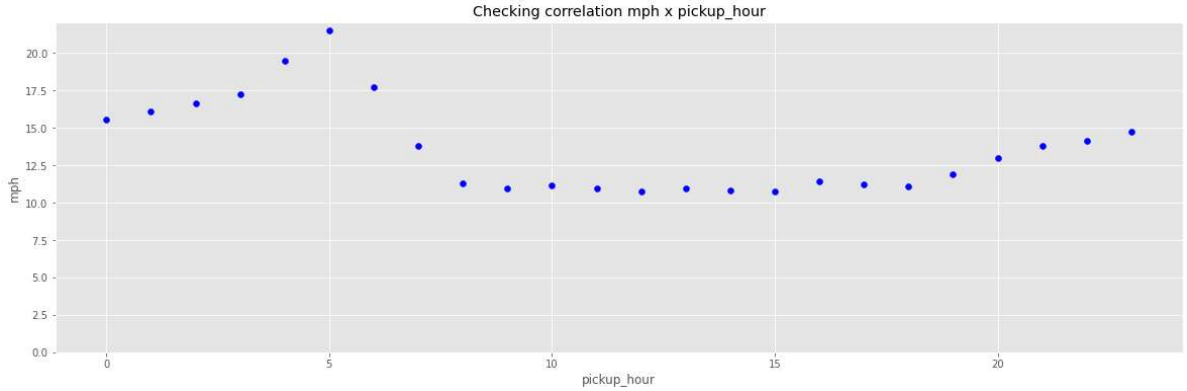
```
df_avg_mph.index
```

Out[32]:

```
Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
16,
            17, 18, 19, 20, 21, 22, 23],
           dtype='int64', name='pickup_hour')

time: 329 ms
```

In [33]:

```
# Checking correlation Avg(Speed) x Hour

df_avg_mph = df[['pickup_hour','mph']].groupby(['pickup_hour']).mean()

x, y = df_avg_mph.index, df_avg_mph['mph']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.ylim(ymin=0)

plt.show()
```

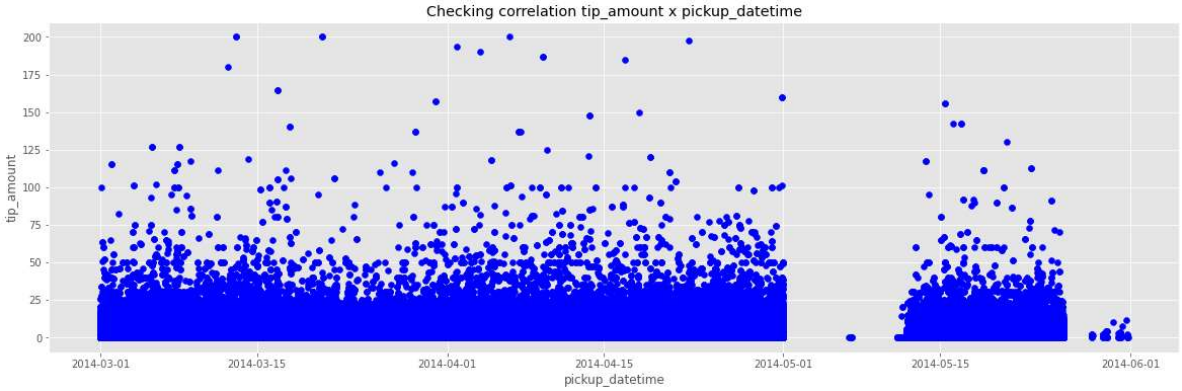



```
time: 20.3 s
```

Target x Predictor: tip_amount (Happiness) *vs.* Pickup datetime

In [25]:

```
# Checking correlation Happiness x Pickup datetime
x, y = df['pickup_datetime'], df['tip_amount']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```

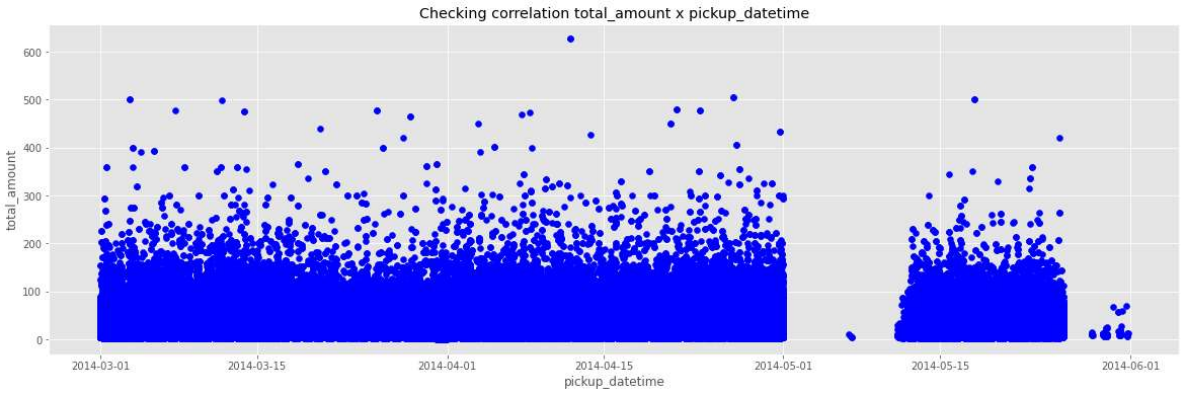


time: 30.2 s

Target x Predictor: total_amount (Revenue) *vs.* Pickup datetime

In [26]:

```
# Checking correlation Dropoff longitude x Pickup longitude
x, y = df['pickup_datetime'], df['total_amount']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```

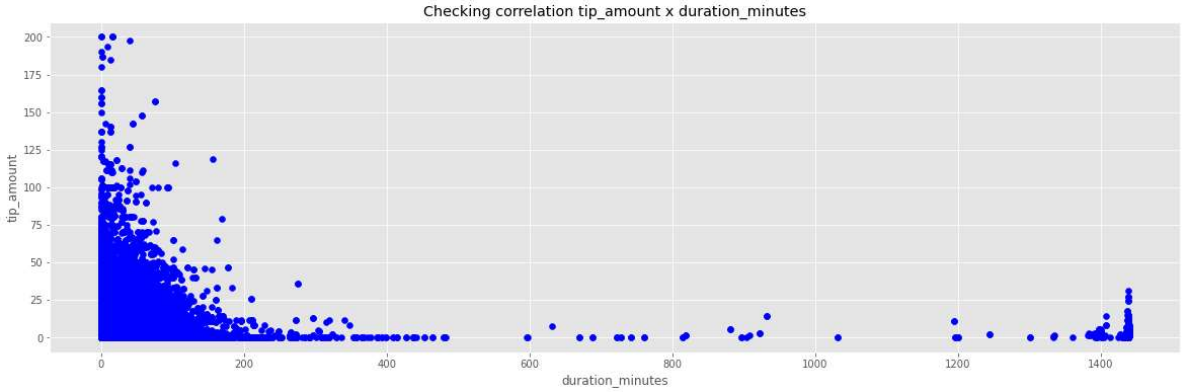


time: 28 s

Target x Predictor: tip_amount (Happiness) *vs.* Trip duration

In [27]:

```
# Checking correlation Happiness x Trip duration
x, y = df['duration_minutes'], df['tip_amount']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```



time: 25.7 s

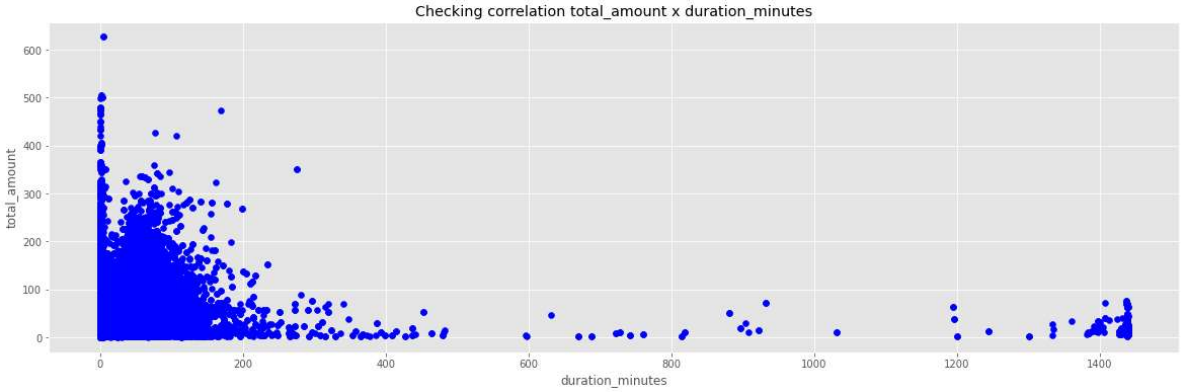
💡 ***Non-linearly correlated. To drivers' usual preference for long trips, the plot adds interesting information:***

➜ ***Skyrocketing tips are exclusive for ultra-short trips.***

Target x Predictor: total_amount (Revenue) *vs.* Trip duration

In [28]:

```
# Checking correlation Dropoff longitude x Trip duration
x, y = df['duration_minutes'], df['total_amount']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```



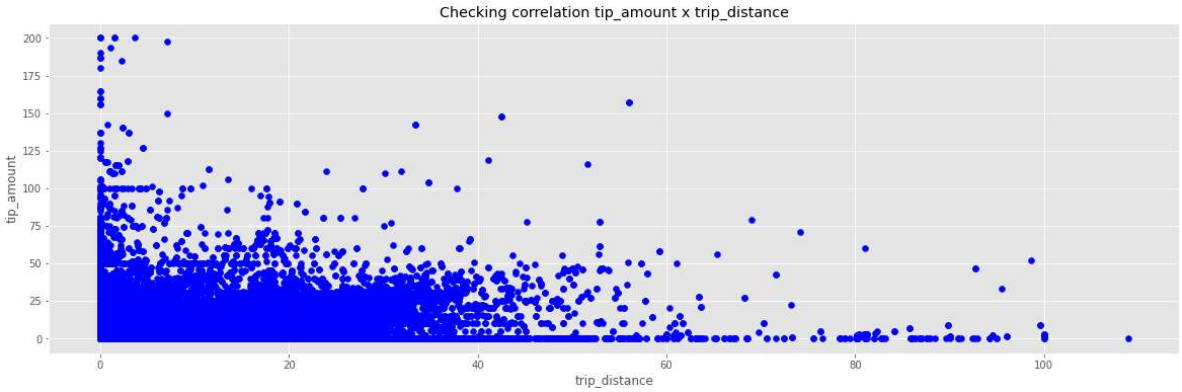
time: 25.7 s

🤔 ***Now things got really strange: are the totals getting lower for higher duration? Weird. Let's take a look at trip distances.***

Target x Predictor: tip_amount (Happiness) *vs.* Trip length

In [29]:

```
# Checking correlation Happiness x Trip length
x, y = df['trip_distance'], df['tip_amount']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```



time: 25.9 s

💡 ***Non-linearly correlated. To drivers' usual preference for long trips, the plot adds interesting***

information:

➜ ***Skyrocketing tips are exclusive for ultra-short trips.***

In [30]:

```
df[df['tip_amount'] > 125 ]
```

Out[30]:

	order_id	passenger_id	pickup_datetime	pickup_month	pickup_week	pickup_v
641648	7075345	5234567822133473	2014-04-07 09:38:00	4	15	
761543	7195240	5234567818262055	2014-04-09 12:34:00	4	15	
1102633	7536330	5234567816761250	2014-04-22 12:24:01	4	17	
1303012	7736709	5234567812369597	2014-05-15 11:29:35	5	20	
1330860	7764557	5234567820640168	2014-05-16 04:19:51	5	20	
1365806	7799503	5234567812377715	2014-05-16 20:45:51	5	20	
2294543	8728240	5234567822103654	2014-03-05 14:49:00	3	10	
3071461	9505158	5234567818084556	2014-03-08 00:44:00	3	10	
3480500	9914197	5234567817990104	2014-03-13 03:31:00	3	11	
3660621	10094318	5234567820590549	2014-03-17 22:53:00	3	12	
3926045	10359742	5234567814040483	2014-03-20 19:07:00	3	12	
4082677	10516374	5234567820242517	2014-03-30 22:15:00	3	13	
4358328	10792025	5234567817867706	2014-03-29 03:35:00	3	13	
4807605	11241302	5234567815438663	2014-03-16 19:45:05	3	11	
6006413	12440110	5234567812363633	2014-04-03 22:24:57	4	14	
6230687	12664384	5234567821680901	2014-04-13 17:38:00	4	15	
6407954	12841651	5234567821030852	2014-04-16 21:13:00	4	16	
7290445	13724142	5234567819186145	2014-04-30 21:57:00	4	18	
7853451	14287148	5234567817656470	2014-04-18 03:36:43	4	16	
8141690	14575387	5234567813738267	2014-04-07 09:38:00	4	15	

	order_id	passenger_id	pickup_datetime	pickup_month	pickup_week	pickup_v
8144824	14578521	5234567812388228	2014-04-07 12:36:00	4	15	
8261538	14695235	5234567823804902	2014-04-09 12:34:00	4	15	
8802979	236676	5234567812376802	2014-05-15 11:29:35	5	20	
8865874	299571	5234567812353561	2014-05-16 20:45:51	5	20	
9165012	598709	5234567812350942	2014-05-20 22:56:57	5	21	
9794506	1228203	5234567815534539	2014-03-05 14:49:00	3	10	
10571442	2005139	5234567818974077	2014-03-08 00:44:00	3	10	
10856659	2290356	5234567812390176	2014-03-12 09:44:00	3	11	
10980503	2414200	5234567816282208	2014-03-13 03:31:00	3	11	
11160631	2594328	5234567820685699	2014-03-17 22:53:00	3	12	
11426040	2859737	5234567813429882	2014-03-20 19:07:00	3	12	
11582683	3016380	5234567813425283	2014-03-30 22:15:00	3	13	
11858328	3292025	5234567817635084	2014-03-29 03:35:00	3	13	
12307646	3741343	5234567814040207	2014-03-16 19:45:05	3	11	
13061116	4494813	5234567812384007	2014-04-01 19:12:00	4	14	
13379249	4812946	5234567823813585	2014-04-06 13:32:00	4	14	
13730689	5164386	5234567814219618	2014-04-13 17:38:00	4	15	
14790445	6224142	5234567817340504	2014-04-30 21:57:00	4	18	

time: 1.38 s

In [31]:

```
print('Weird ultra-short trips with ultra-high tips:')
print('# trips: '     ,     (df.loc[df['tip_amount'] > 125, 'duration_minutes']).count())
print('avg duration: ', int(round(df.loc[df['tip_amount'] > 125, 'duration_minutes'].mean()
print('avg distance: ', int(round(df.loc[df['tip_amount'] > 125, 'trip_distance'].mean(),0)
print('avg tip: '     , int(round(df.loc[df['tip_amount'] > 125, 'tip_amount'].mean(),0)))
```

```
Weird ultra-short trips with ultra-high tips:
# trips:  38
avg duration:  16
avg distance:  8
avg tip:  160
time: 312 ms
```

In [32]:

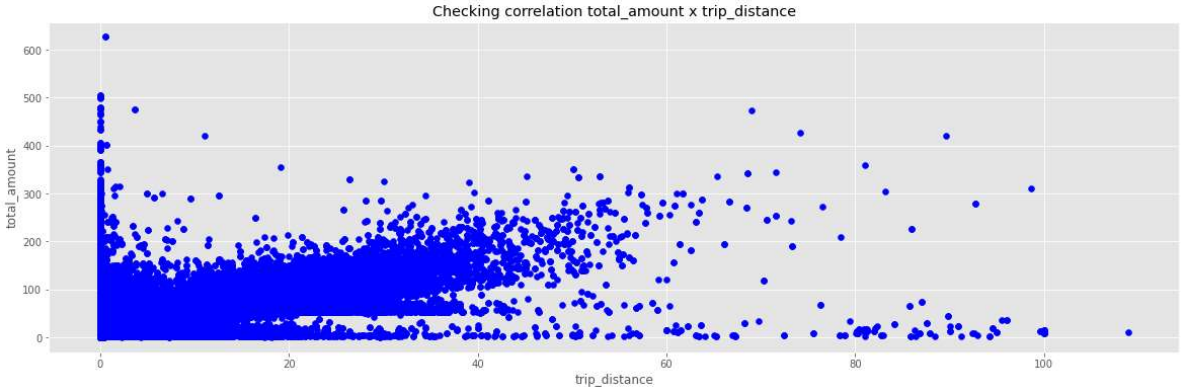
```
print('Weird ultra-high tips for zero time:')
print('# trips: '     ,     (df.loc[(df['tip_amount'] > 125) & (df['duration_minutes'] == 0
print('avg duration: ', int(round(df.loc[(df['tip_amount'] > 125) & (df['duration_minutes']
print('avg distance: ', int(round(df.loc[(df['tip_amount'] > 125) & (df['duration_minutes']
print('avg tip: '     , int(round(df.loc[(df['tip_amount'] > 125) & (df['duration_minutes']
```

```
Weird ultra-high tips for zero time:
# trips:  10
avg duration:  0
avg distance:  0
avg tip:  156
time: 547 ms
```

Target x Predictor: total_amount (Revenue) *vs.* Trip length

In [33]:

```
# Checking correlation Dropoff longitude x Trip length
x, y = df['trip_distance'], df['total_amount']

plt.figure(figsize=(20, 6))
plt.scatter(x, y, c='blue')
plt.title('Checking correlation '+ y.name + ' x '+ x.name)
plt.xlabel(x.name)
plt.ylabel(y.name)
plt.show()
```



time: 26.1 s

In [34]:

```
threshold = 300
print('Weird ultra-short trips with ultra-high total amount:')
print('# trips: '     ,          (df.loc[df['total_amount'] > threshold, 'duration_minutes'
print('avg duration: ', int(round(df.loc[df['total_amount'] > threshold, 'duration_minutes'
print('avg distance: ', int(round(df.loc[df['total_amount'] > threshold, 'trip_distance'].m
print('avg total: '    , int(round(df.loc[df['total_amount'] > threshold, 'total_amount'].me
```

```
Weird ultra-short trips with ultra-high total amount:
# trips:  133
avg duration:  20
avg distance:  12
avg total:  372
time: 125 ms
```

In [35]:

```
threshold = 300
print('Weird ultra-high tips for zero time:')
print('# trips: '     ,          (df.loc[(df['total_amount'] > threshold) & (df['duration_m
print('avg duration: ', int(round(df.loc[(df['total_amount'] > threshold) & (df['duration_m
print('avg distance: ', int(round(df.loc[(df['total_amount'] > threshold) & (df['duration_m
print('avg tip: '     , int(round(df.loc[(df['total_amount'] > threshold) & (df['duration_m
```

```
Weird ultra-high tips for zero time:
# trips:  56
avg duration:  0
avg distance:  0
avg tip:  382
time: 531 ms
```

🤔 ***Now things got really strange: are the totals getting lower with time? Weird. Let's take a look at trip***

distances.

Exploratory Data Analysis • Aggregations: frequency distributions

Trips: Frequency distribution by month (%)

In [36]:

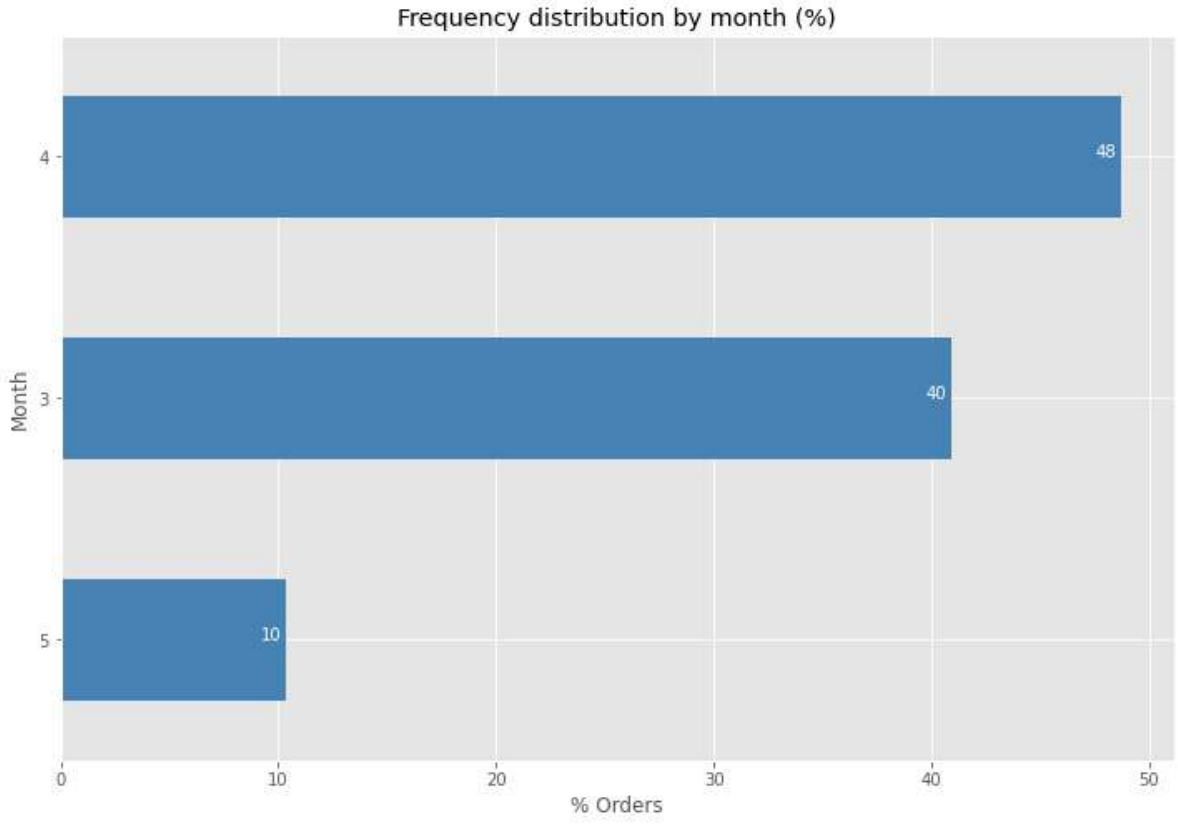
```
orders_by_month = 100 * df['pickup_month'].value_counts() / len(df)
orders_by_month
```

Out[36]:

```
4    48.729219
3    40.914683
5    10.356097
Name: pickup_month, dtype: float64

time: 468 ms
```

In [37]:

```
## Distribution %

orders_by_month.sort_values(ascending=True, inplace=True)
orders_by_month.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Orders')
plt.ylabel('Month')
plt.title('Frequency distribution by month (%)')

for i, value in enumerate(orders_by_month):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 3% from x, and 0 from y to make it fit with
    plt.annotate(label, xy=(value - 1.2, i - 0), color='white')

plt.show()
```



time: 2.02 s

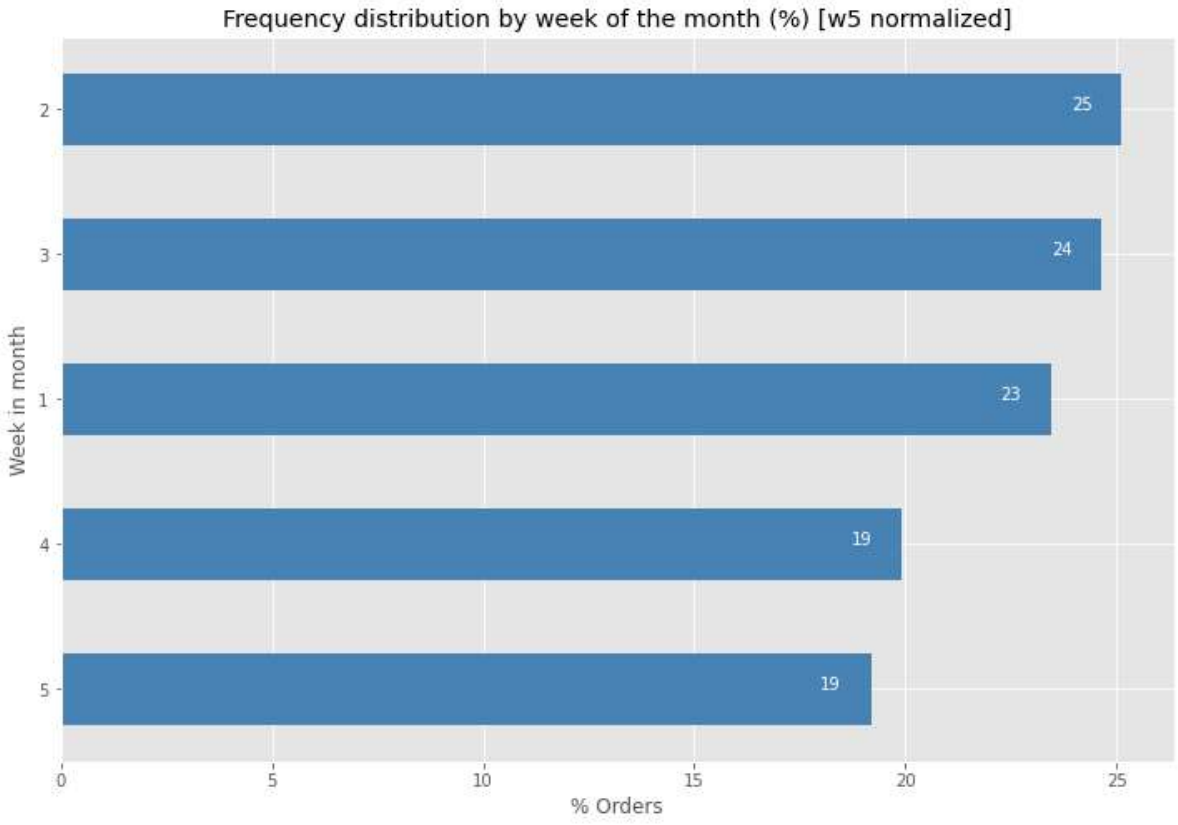
⚠ *There seems to be something wrong with data for May, or there was an extreme event hindering taxi-hailing in the period.* ***Qualitative input needed here.*** *As mentioned above (Predictor x Predictor: Dropoff datetime vs. Pickup datetime), there were quarantines given to Ebola in 2014 (but, in a quick assess, they seem to have occured later). To be checked.*

Trips: Frequency distribution by week of the month (%) [normalized]

In [38]:

```
## Distribution %

days_w5 = 3 * (0.40914707 + 0.10356093) + 2 * (0.48729200) # as seen in 99__DataTransformat
print(' Average days in week 5:', days_w5, '\n')
orders_by_weekinmonth = 100 * df['pickup_weekinmonth'].value_counts() / len(df) * [1, 1, 1,

orders_by_weekinmonth.sort_values(ascending=True, inplace=True)
orders_by_weekinmonth.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Orders')
plt.ylabel('Week in month')
plt.title('Frequency distribution by week of the month (%) [w5 normalized]')

for i, value in enumerate(orders_by_weekinmonth):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 1.2% from x, and 0 from y to make it fit wi
    plt.annotate(label, xy=(value - 1.2, i - 0), color='white')

plt.show()
```

```
 Average days in week 5: 2.512708
```



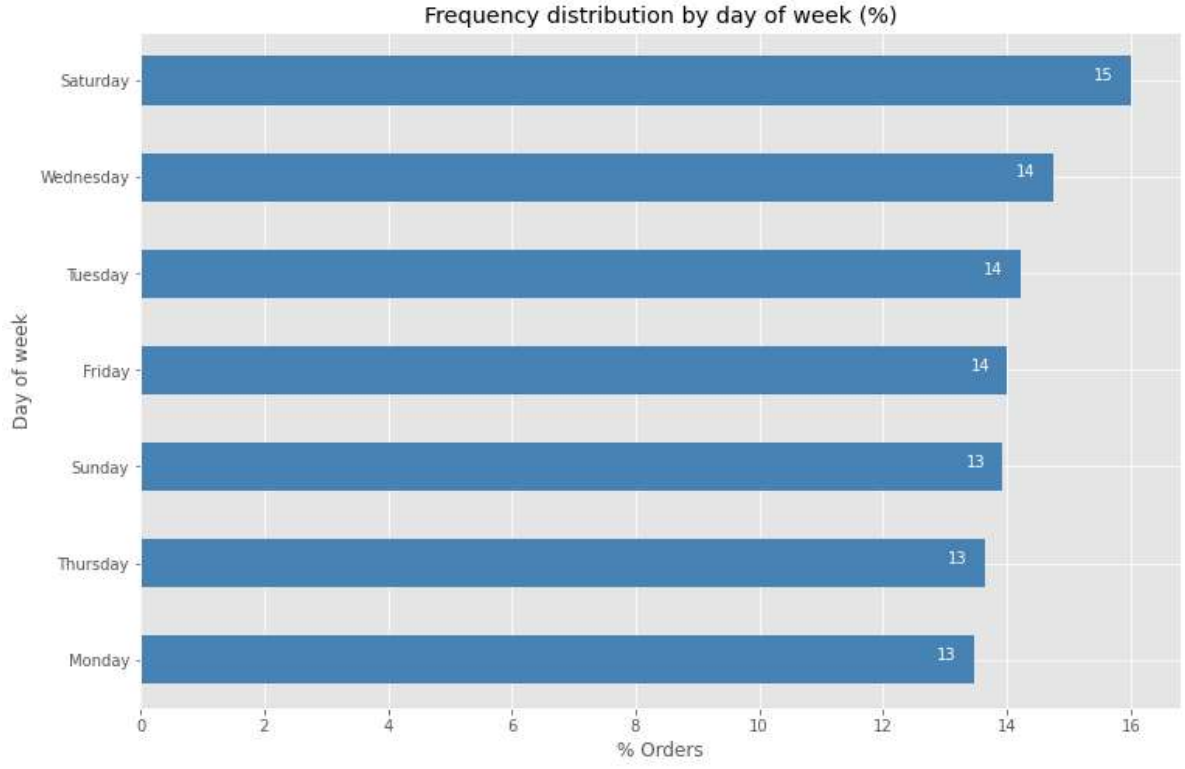

```
time: 1.91 s
```

There is a lower demand at the end of the month.

Trips: Frequency distribution by day of week (%)

In [39]:

```
## Distribution %

orders_by_dayofweek = 100 * df['pickup_dayofweek'].value_counts() / len(df)

orders_by_dayofweek.sort_values(ascending=True, inplace=True)
orders_by_dayofweek.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Orders')
plt.ylabel('Day of week')
plt.title('Frequency distribution by day of week (%)')

for i, value in enumerate(orders_by_dayofweek):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 0.8% from x, and 0 from y to make it fit wi
    plt.annotate(label, xy=(value - 0.6, i - 0), color='white')

plt.show()
```




```
time: 2.12 s
```

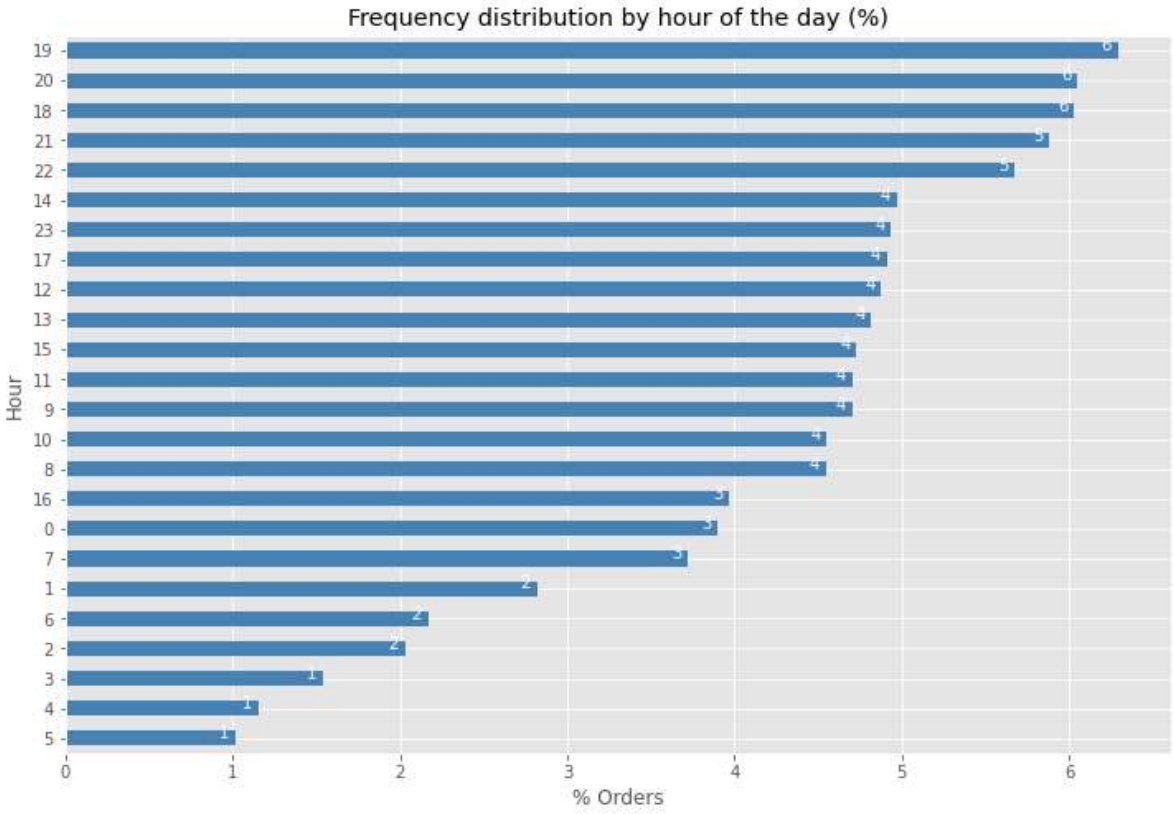
💡 ***Highest** demand on **Saturday**.* 💡 ***Lowest** demand on **Monday**.*

The pattern indicates a predominance of leisure demand, with some 'compensation feelings' when the workweek starts, for the extra spending on the weekend.

Trips: Frequency distribution by pickup hour (%)

In [40]:

```
## Distribution %

orders_by_hour = 100 * df['pickup_hour'].value_counts() / len(df)

orders_by_hour.sort_values(ascending=True, inplace=True)
orders_by_hour.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Orders')
plt.ylabel('Hour')
plt.title('Frequency distribution by hour of the day (%)')

for i, value in enumerate(orders_by_hour):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 0.8% from x, and 0 from y to make it fit wi
    plt.annotate(label, xy=(value - 0.1, i - 0), color='white')

plt.show()
```



time: 1.92 s

💡 ***Highest* demand from *18:00 to 23:00*.** 💡 ***Lowest* demand from *01:00 to 07:00*.**

💡 ***Preferred hours indicate that passengers are more inclined to go home 🏠 by taxi (rather than take taxi to get to work faster/rested, e.g.).***

With some aggregation, we can heve a clearer view. Let's create a new feature: time of the day.

Trips: Frequency distribution by pickup time of the day (%)

In [35]:

```
df.insert(loc=10, column='pickup_timeofday', value = df['pickup_hour'])
```

time: 390 ms

In [36]:

```
timeofday_dict = {
                23: 'Night [20:00 to 01:00]',
                20: 'Evening [17:00 to 20:00]',
                17: 'Afternoon [12:00 to 17:00]',
                12: 'Morning [08:00 to 12:00]',
                 8: 'Early morning [05:00 to 08:00]',
                 5: 'Middle of the night [01:00 to 05:00]',
                 1: 'Night [20:00 to 01:00]',
                }
```

time: 62 ms

In [37]:

```
for i in range(len(timeofday_dict)):
    df.loc[df['pickup_hour'] <= int(list(timeofday_dict)[i]), 'pickup_timeofday'] = list(ti
df.head(1)
```

Out[37]:

	order_id	passenger_id	pickup_datetime	pickup_month	pickup_week	pickup_weekinmo
0	6433697	5234567816269547	2014-04-11 18:06:45	4	15	

time: 6min 41s

In [39]:

```
df.count()
```

Out[39]:

```
order_id               14999994
passenger_id           14999994
pickup_datetime        14999994
pickup_month           14999994
pickup_week            14999994
pickup_weekinmonth     14999994
pickup_date            14999994
pickup_dayofweek       14999994
pickup_isweekend       14999994
pickup_hour            14999994
pickup_timeofday       14999994
pickup_isrush          14999994
pickup_latitude        14726894
pickup_longitude       14726888
dropoff_latitude       11237187
dropoff_longitude      11237178
dropoff_datetime       11456221
duration               11456221
duration_minutes       11456221
duration_type          11456221
trip_distance          11456221
total_amount           11456221
fare_amount            11456221
tip_amount             11456221
rate_type              11456221
payment_type           14999994
pickup_borough         14999994
dropoff_borough        11456221
mph                    11381250
dtype: int64


time: 11 s
```

In [68]:

```
orders_by_timeofday = 100 * df['pickup_timeofday'].value_counts() / len(df['pickup_timeofda
orders_by_timeofday
```

Out[68]:

```
Afternoon [12:00 to 17:00]              23.400716
Night [20:00 to 01:00]                  23.212143
Morning [08:00 to 12:00]                18.839088
Evening [17:00 to 20:00]                18.370234
Early morning [05:00 to 08:00]          10.440578
Middle of the night [01:00 to 05:00]     5.737242
Name: pickup_timeofday, dtype: float64

time: 1.55 s
```

In [69]:

```
sum(orders_by_timeofday)
```

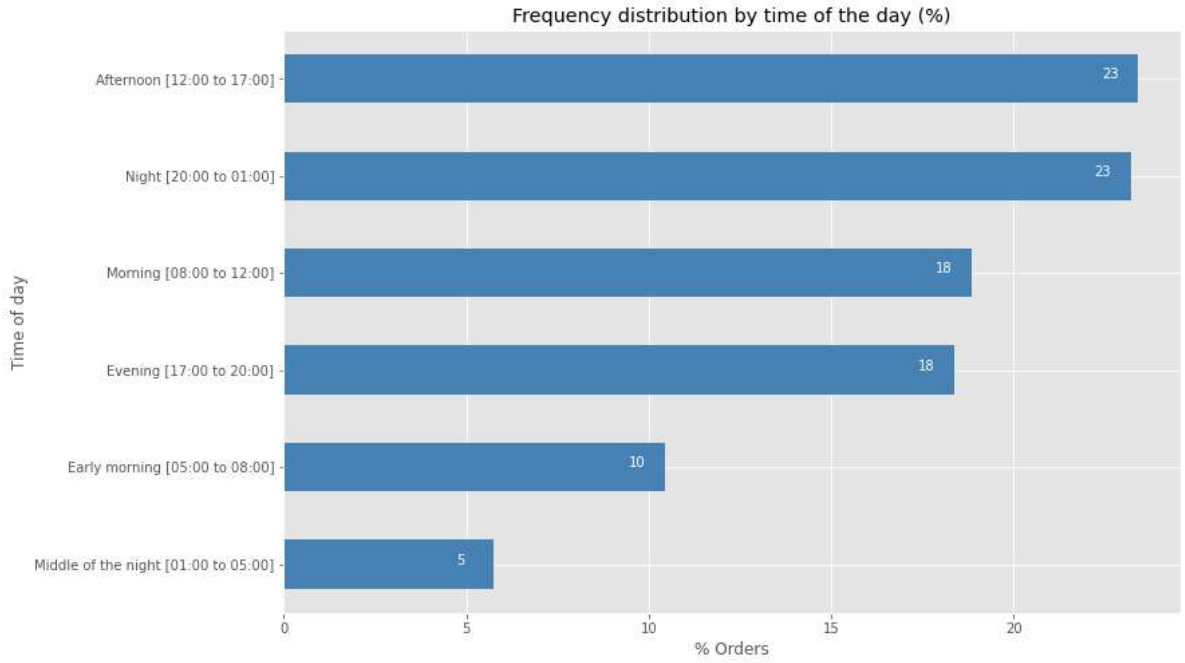
Out[69]:

```
100.0
```

```
time: 0 ns
```

In [70]:

```
## Distribution %

orders_by_timeofday = 100 * df['pickup_timeofday'].value_counts() / len(df)

orders_by_timeofday.sort_values(ascending=True, inplace=True)
orders_by_timeofday.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Orders')
plt.ylabel('Time of day')
plt.title('Frequency distribution by time of the day (%)')

for i, value in enumerate(orders_by_timeofday):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 0.8% from x, and 0 from y to make it fit wi
    plt.annotate(label, xy=(value - 1, i - 0), color='white')

plt.show()
```




```
time: 3.19 s
```

💡 ***Highest demand on Afternoon [12:00 to 17:00].*** 💡 ***Lowest demand on Middle of the night [01:00 to 05:00].***

A better distribution can be seen by taking hourly data (normalizing for the different number of hours in each bin of time of the day).

Trips: Frequency distribution by pickup time of the day (% hourly

demand)

In [95]:

```
orders_by_timeofday = 100 * df['pickup_timeofday'].value_counts() / len(df['pickup_timeofda
orders_by_timeofday
```

Out[95]:

```
Afternoon [12:00 to 17:00]              23.400716
Night [20:00 to 01:00]                  23.212143
Morning [08:00 to 12:00]                18.839088
Evening [17:00 to 20:00]                18.370234
Early morning [05:00 to 08:00]          10.440578
Middle of the night [01:00 to 05:00]     5.737242
Name: pickup_timeofday, dtype: float64

time: 1.39 s
```

In [100]:

```
#normalizing
orders_by_timeofday_nrm = orders_by_timeofday / [5,5,4,3,3,4]
orders_by_timeofday_nrm /= sum(orders_by_timeofday_nrm)
orders_by_timeofday_nrm *= 100
orders_by_timeofday_nrm.sort_values(ascending=False, inplace=True)
orders_by_timeofday_nrm
```

Out[100]:

```
Evening [17:00 to 20:00]                24.425003
Morning [08:00 to 12:00]                18.786292
Afternoon [12:00 to 17:00]              18.668109
Night [20:00 to 01:00]                  18.517673
Early morning [05:00 to 08:00]          13.881758
Middle of the night [01:00 to 05:00]     5.721164
Name: pickup_timeofday, dtype: float64

time: 15 ms
```

In [101]:

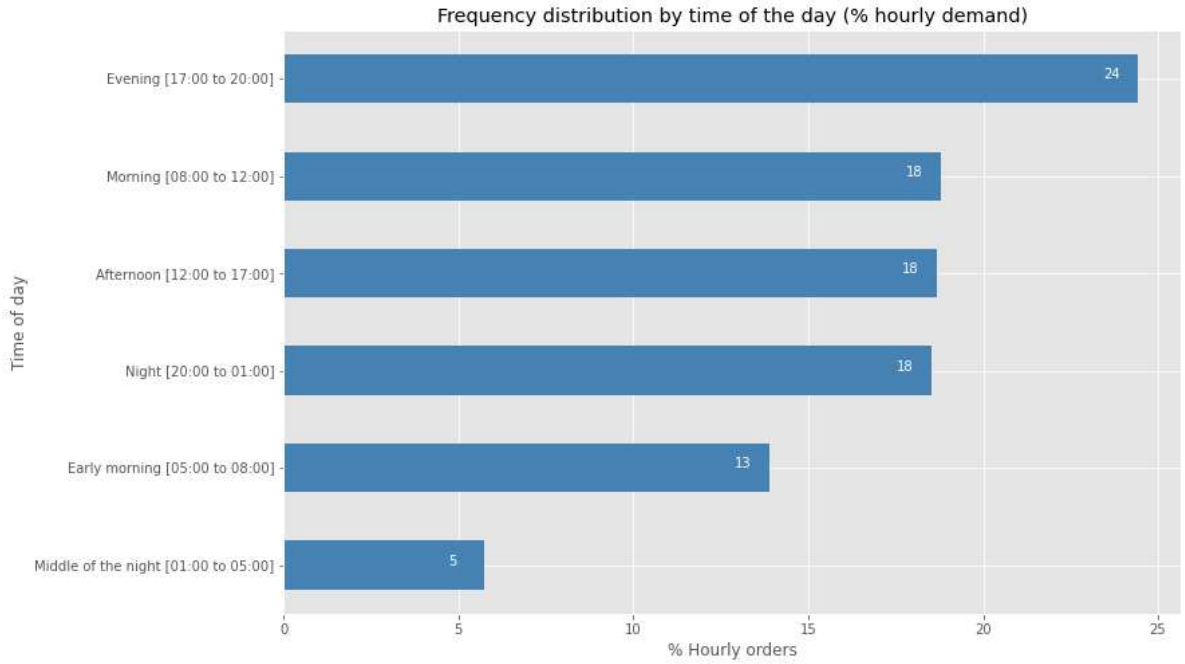
```
sum(orders_by_timeofday_nrm)
```

Out[101]:

```
100.00000000000001

time: 0 ns
```

In [102]:

```
## Distribution %

orders_by_timeofday_nrm.sort_values(ascending=True, inplace=True)
orders_by_timeofday_nrm.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Hourly orders')
plt.ylabel('Time of day')
plt.title('Frequency distribution by time of the day (% hourly demand)')

for i, value in enumerate(orders_by_timeofday_nrm):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 1% from x, and 0 from y to make it fit with
    plt.annotate(label, xy=(value - 1, i - 0), color='white')

plt.show()
```



time: 297 ms

💡 ***OPERATIONS:***

☺ *There is a nice baseline demand on Mornings, Nights, Afternoons.*

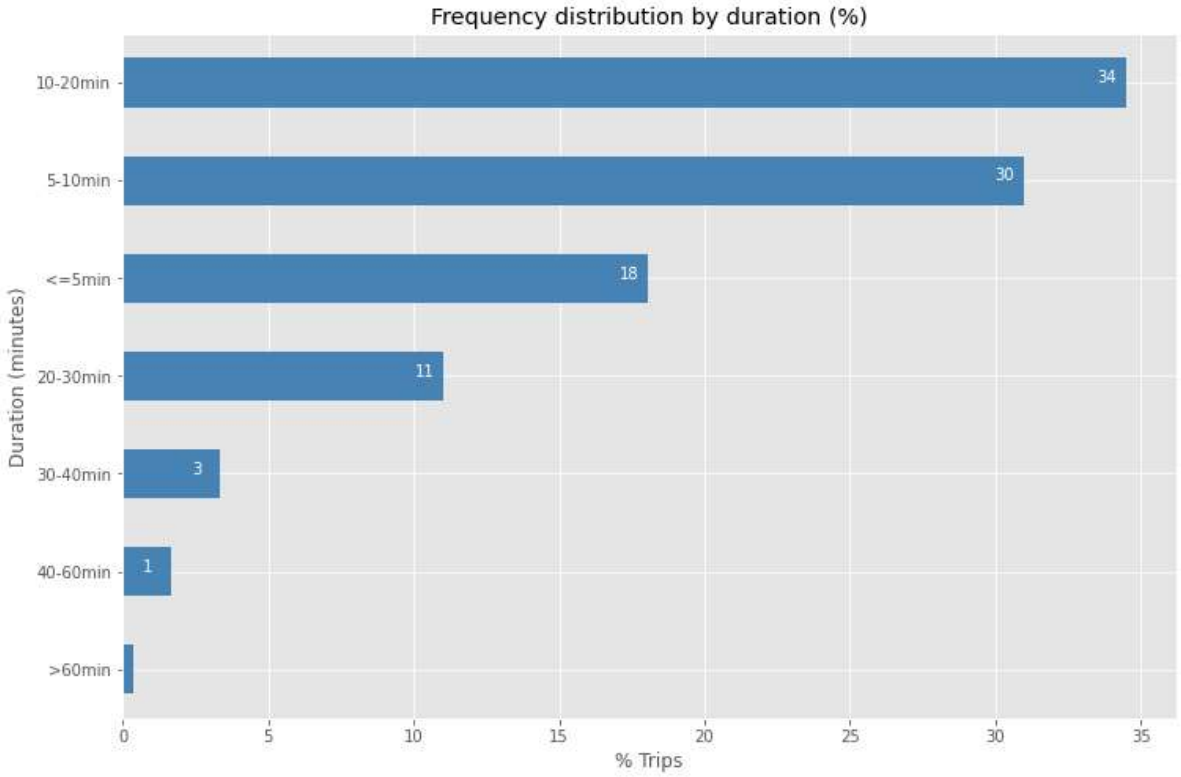
🗲 *Peak demand occurs on Evenings.*

✅ *Low demand takes place on Early mornings and in the MIddle of the night.*

Trips: Frequency distribution by trajectory duration (%)

In [73]:

```
## Distribution %

orders_by_duration = 100 * df['duration_type'].value_counts() / (df['duration_type'].count(

orders_by_duration.sort_values(ascending=True, inplace=True)
orders_by_duration.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Trips')
plt.ylabel('Duration (minutes)')
plt.title('Frequency distribution by duration (%)')

for i, value in enumerate(orders_by_duration):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 0.8% from x, and 0 from y to make it fit wi
    plt.annotate(label, xy=(value - 1, i - 0), color='white')

plt.show()
```



time: 3.39 s

💡 ***82% of trips take 20 min or less***

(Resuming after Kernel interruption)

In [5]:

```
with open('df_orders_tF_EDA.csv') as f: cols = f.readline().split(',')
del cols[0]
cols[-1] = cols[-1].replace('\n','')
df_original = pd.read_csv('df_orders_tF_EDA.csv', usecols=cols) # reading previously transf
df = pd.read_csv('df_orders_tF_EDA.csv', usecols=cols) # reading previously transformed dat
df.head(2)
```

Out[5]:

	order_id	passenger_id	pickup_datetime	pickup_month	pickup_week	pickup_weekinmo
0	6433697	5234567816269547	2014-04-11 18:06:45	4	15	
1	6433698	5234567812367422	2014-04-12 00:20:28	4	15	

time: 10min 53s

In [51]:

```
%matplotlib inline

import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.style.use('ggplot')
```

time: 0 ns

Trips: Frequency distribution by trajectory length (%)

In [7]:

```
df['trip_distance'].describe()
```

Out[7]:

```
count    1.145622e+07
mean     2.903006e+00
std      3.423146e+00
min      0.000000e+00
25%      1.020000e+00
50%      1.770000e+00
75%      3.200000e+00
max      1.089000e+02
Name: trip_distance, dtype: float64
```

time: 1.14 s

In [56]:

```
print(df['trip_distance'].describe()[7])
print(df['trip_distance'].describe()[6])
print(df['trip_distance'].describe()[5])
print(df['trip_distance'].describe()[4])
```

```
108.9
3.2
1.77
1.02
time: 3.52 s
```

In [78]:

```
orders_by_distance_dict = {'far (more than 10 miles)':              100 * sum( (df['trip_di
orders_by_distance_dict['moderately distant (more than 5 miles)'] = 100 * sum( (df['trip_di
orders_by_distance_dict['close (1 to 5 miles)']                   = 100 * sum( (df['trip_di
orders_by_distance_dict['very close (less than 1 mile)']          = 100 * sum( (df['trip_di
orders_by_distance_dict
```

Out[78]:

```
{'far (more than 10 miles)': 4.901302096040221,
 'moderately distant (more than 5 miles)': 8.76107400511914,
 'close (1 to 5 miles)': 61.755241977262834,
 'very close (less than 1 mile)': 24.582381921577806}

time: 8.86 s
```

In [65]:

```
sum(orders_by_distance_dict.values())
```

Out[65]:

```
100.0

time: 0 ns
```

In [67]:

```
orders_by_distance = pd.DataFrame(
                       orders_by_distance_dict.values(),
             index = orders_by_distance_dict.keys(),
             columns = ['%'])
orders_by_distance.sort_values(by='%',ascending=False, inplace=True)
orders_by_distance
```

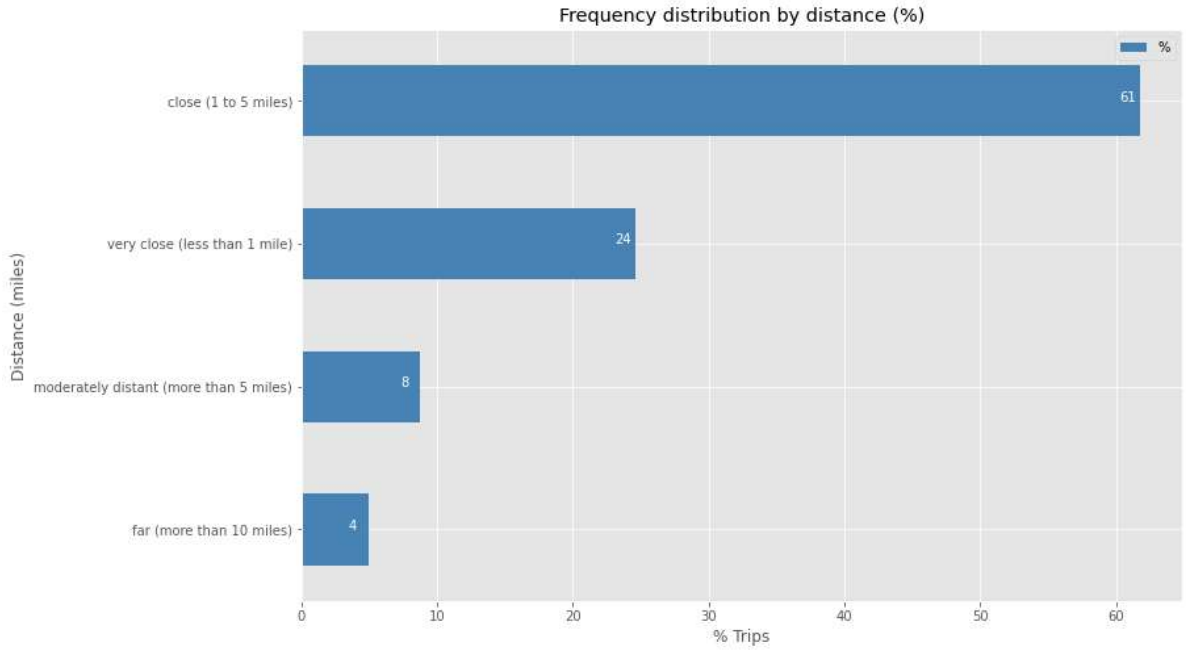
Out[67]:

	%
close (1 to 5 miles)	61.755242
very close (less than 1 mile)	24.582382
moderately distant (more than 5 miles)	8.761074
far (more than 10 miles)	4.901302

```
time: 0 ns
```

In [70]:

```
## Distribution %

orders_by_distance.sort_values(by='%',ascending=True, inplace=True)
orders_by_distance.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Trips')
plt.ylabel('Distance (miles)')
plt.title('Frequency distribution by distance (%)')

for i, value in enumerate(orders_by_distance['%']):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 1% from x, and 0 from y to make it fit with
    plt.annotate(label, xy=(value - 1.5, i - 0), color='white')

plt.show()
```



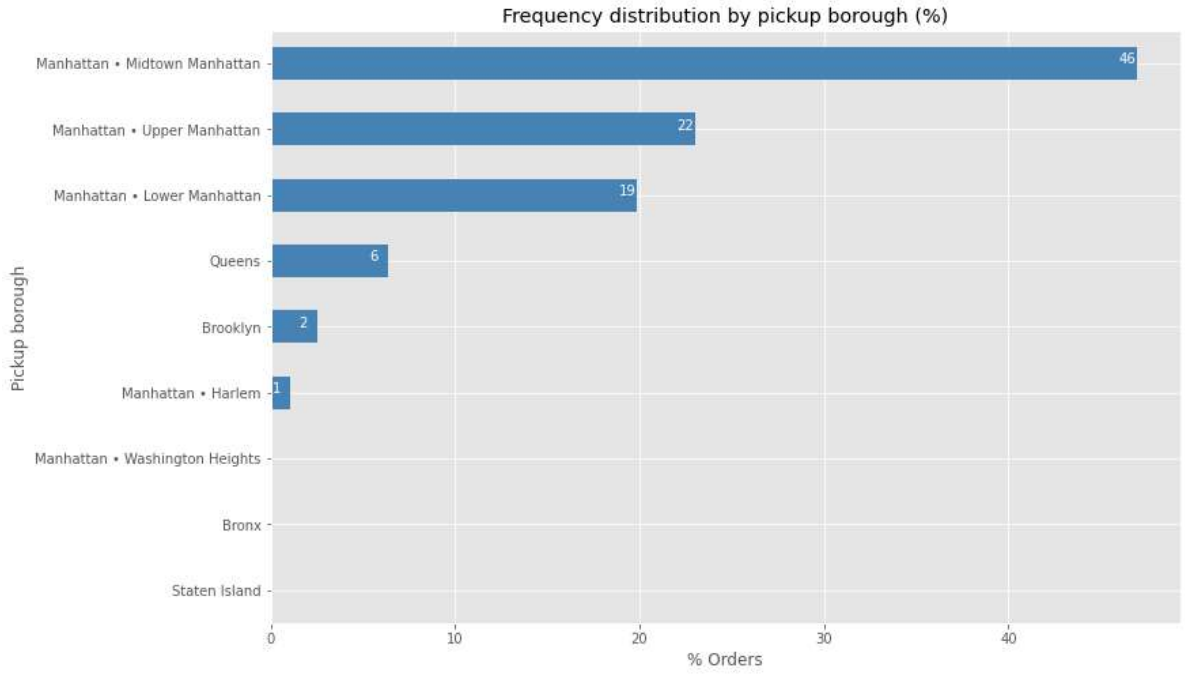
time: 391 ms

💡 ***85% of trips are to close places, up to 5 miles distant.*** 💡 ***24% take very close, less than 1 mile away.***

Trips: Frequency distribution by pickup borough (%)

In [132]:

```
## Distribution %

orders_by_pickup_borough = 100 * df['pickup_borough'].value_counts() / (df['pickup_borough'

orders_by_pickup_borough.sort_values(ascending=True, inplace=True)
orders_by_pickup_borough.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Orders')
plt.ylabel('Pickup borough')
plt.title('Frequency distribution by pickup borough (%)')

for i, value in enumerate(orders_by_pickup_borough):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 1% from x, and 0 from y to make it fit with
    plt.annotate(label, xy=(value - 1, i - 0), color='white')

plt.show()
```

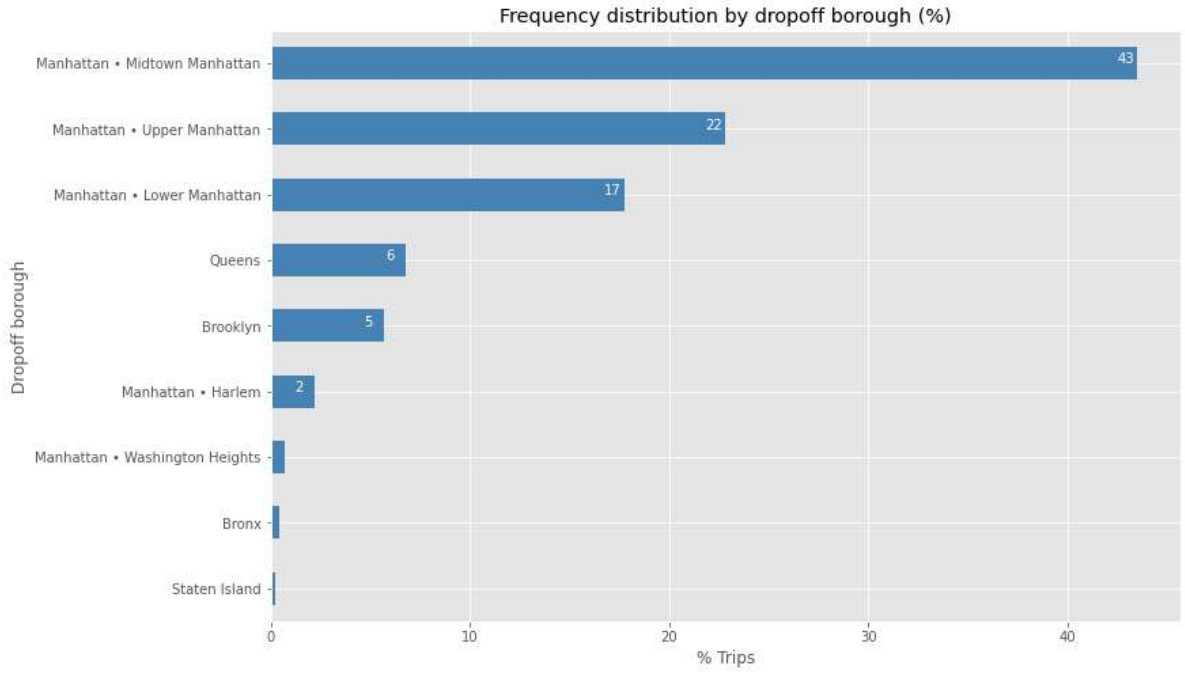


time: 4.45 s

Trips: Frequency distribution by dropoff borough (%)

In [71]:

```
## Distribution %

orders_by_dropoff_borough = 100 * df['dropoff_borough'].value_counts() / (df['dropoff_borou

orders_by_dropoff_borough.sort_values(ascending=True, inplace=True)
orders_by_dropoff_borough.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Trips')
plt.ylabel('Dropoff borough')
plt.title('Frequency distribution by dropoff borough (%)')

for i, value in enumerate(orders_by_dropoff_borough):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 1% from x, and 0 from y to make it fit with
    plt.annotate(label, xy=(value - 1, i - 0), color='white')

plt.show()
```



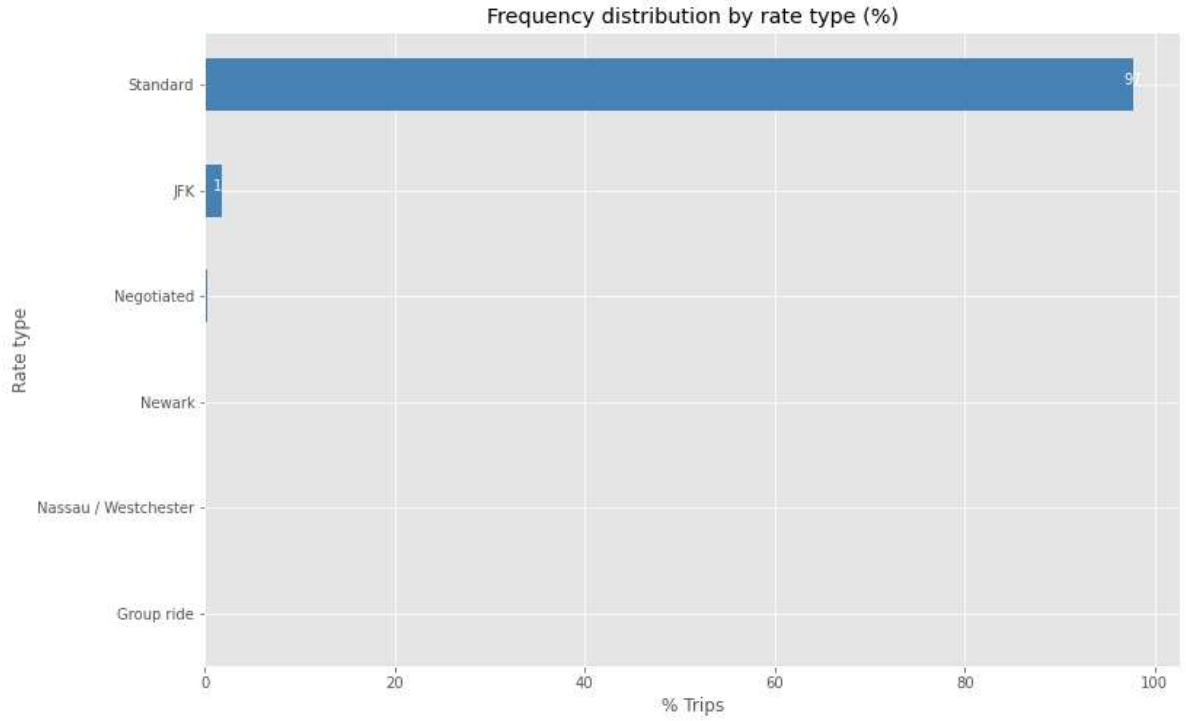
time: 3.23 s

💡 ***Meeting the inference we got for the chart of distribution for pickup hour, the pattern with pickup/dropoff boroughs indicates that passengers are <u>more inclined to go home</u> 🏠 by taxi (rather than take taxi to get to work faster/rested, e.g.).***

TRIPS: Frequency distribution by rate type (%)

In [72]:

```
## Distribution %

orders_by_rate_type = 100 * df['rate_type'].value_counts() / (df['rate_type'].count())

orders_by_rate_type.sort_values(ascending=True, inplace=True)
orders_by_rate_type.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Trips')
plt.ylabel('Rate type')
plt.title('Frequency distribution by rate type (%)')

for i, value in enumerate(orders_by_rate_type):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 0.8% from x, and 0 from y to make it fit wi
    plt.annotate(label, xy=(value - 1, i - 0), color='white')

plt.show()
```



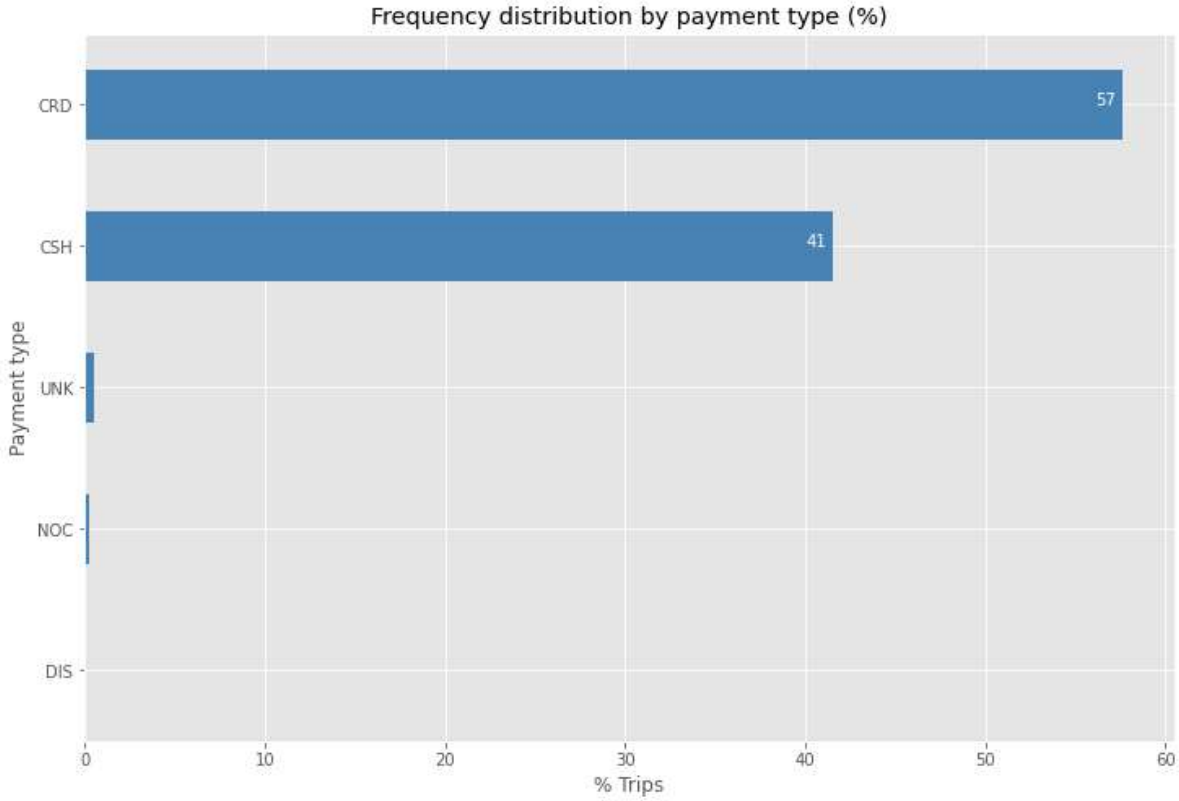
time: 2.88 s

💡 ***Non-standard rate represents an ultra-tiny fraction of the business.*** 💡 ***Exception for the 1% of JFK.***

TRIPS: Frequency distribution by payment type (%)

In [74]:

```
## Distribution %

orders_by_payment_type = 100 * df['payment_type'].value_counts() / (df['payment_type'].coun

orders_by_payment_type.sort_values(ascending=True, inplace=True)
orders_by_payment_type.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Trips')
plt.ylabel('Payment type')
plt.title('Frequency distribution by payment type (%)')

for i, value in enumerate(orders_by_payment_type):
    label = format(int(value), ',') # format int with commas

    # place text at the end of bar (subtracting 0.8% from x, and 0 from y to make it fit wi
    plt.annotate(label, xy=(value - 1.5, i - 0), color='white')

plt.show()
```



time: 2.75 s

PAYMENTS: 98% card/cash

💡 ***57% of trips are paid by card.*** 💡 ***41% of trips are paid in cash.***

TRIPS: Frequency distribution by tip amount (%)

In [75]:

```
df['tip_amount'].describe()
```

Out[75]:

```
count    1.145622e+07
mean     1.481081e+00
std      2.276245e+00
min      0.000000e+00
25%      0.000000e+00
50%      1.000000e+00
75%      2.000000e+00
max      2.000000e+02
Name: tip_amount, dtype: float64

time: 1.05 s
```

In [130]:

```
print(df['tip_amount'].describe()[7])
print(df['tip_amount'].describe()[6])
print(df['tip_amount'].describe()[5])
print(df['tip_amount'].describe()[4])
```

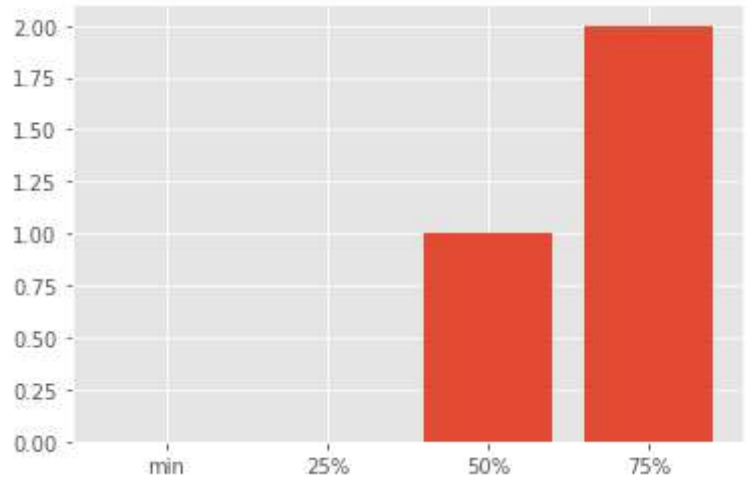
```
200.0
2.0
1.0
0.0
time: 3.23 s
```

In [104]:

```
plt.bar(df['tip_amount'].describe().keys()[3:7], df['tip_amount'].describe()[3:7])
```

Out[104]:

```
<BarContainer object of 4 artists>
```




```
time: 1.83 s
```

💡 ***The typical tip for taxi trips in New York Mar-May 2014 has been $1.***

In [125]:

```
orders_by_tip_amount_dict = {'very generous ($10+)': 100 * sum( (df['tip_amount'] <= 200) &
orders_by_tip_amount_dict['generous ($1+)']        = 100 * sum( (df['tip_amount'] <=   2) &
orders_by_tip_amount_dict['typical ($1)']          = 100 * sum( (df['tip_amount'] ==   1)
orders_by_tip_amount_dict['low (some cents)']      = 100 * sum( (df['tip_amount']  <   1) &
orders_by_tip_amount_dict['no tip']                = 100 * sum( (df['tip_amount'] ==   0)
orders_by_tip_amount_dict
```

Out[125]:

```
{'very generous ($10+)': 24.234780387005454,
 'generous ($1+)': 21.185319312537704,
 'typical ($1)': 9.005814395514891,
 'low (some cents)': 1.759515637835548,
 'no tip': 43.814570267106404}

time: 10.8 s
```

In [126]:

```
sum(orders_by_tip_amount_dict.values())
```

Out[126]:

```
100.0

time: 0 ns
```

In [127]:

```
orders_by_tip_amount = pd.DataFrame(
                       orders_by_tip_amount_dict.values(),
              index = orders_by_tip_amount_dict.keys(),
              columns = ['%'])
orders_by_tip_amount.sort_values(by='%',ascending=False, inplace=True)
orders_by_tip_amount
```

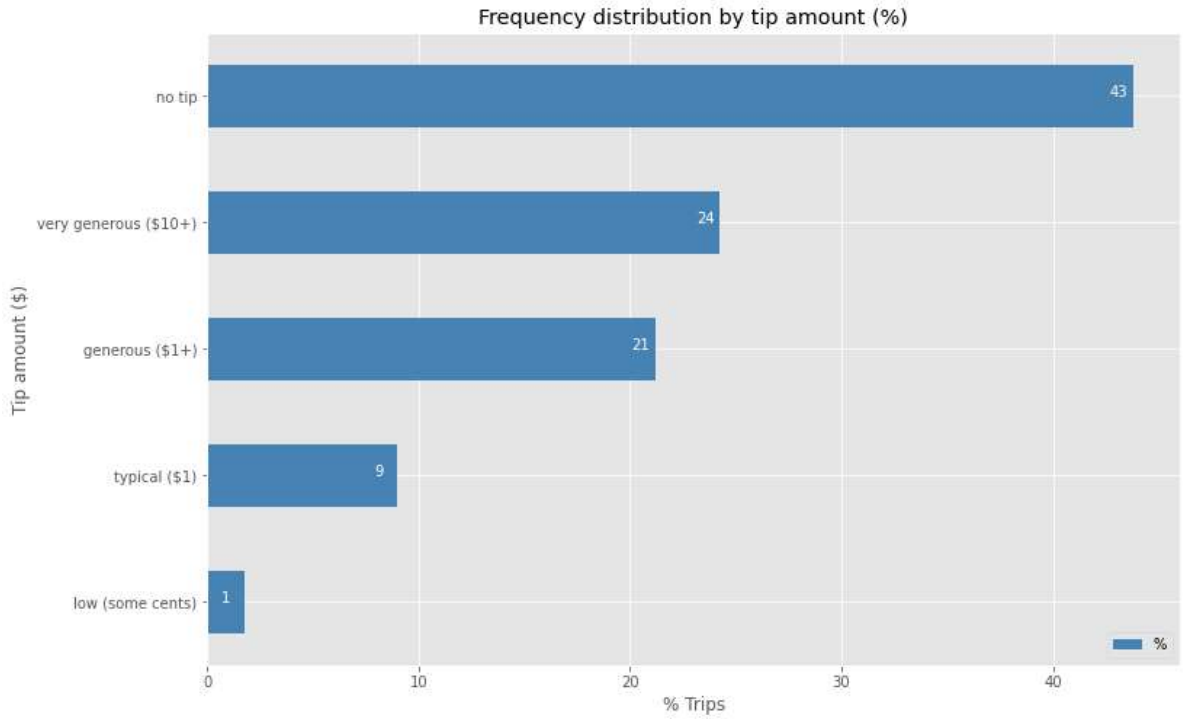
Out[127]:

	%
no tip	43.814570
very generous ($10+)	24.234780
generous ($1+)	21.185319
typical ($1)	9.005814
low (some cents)	1.759516

```
time: 141 ms
```

In [128]:

```
## Distribution %

orders_by_tip_amount.sort_values(by='%',ascending=True, inplace=True)
orders_by_tip_amount.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Trips')
plt.ylabel('Tip amount ($)')
plt.title('Frequency distribution by tip amount (%)')

for i, value in enumerate(orders_by_tip_amount['%']):
    label = format(int(value))

    # place text at the end of bar (subtracting 1% from x, and 0 from y to make it fit with
    plt.annotate(label, xy=(value - 1.1, i - 0), color='white')

plt.show()
```



time: 375 ms

💡 ***Typical tip: $1.***

💡 ***57% of trips have been tipped.*** 💡 ***83% of them above the typical amount.***

Trips: Frequency distribution by total amount (%)

In [109]:

```
df['total_amount'].describe()
```

Out[109]:

```
count    1.145622e+07
mean     1.499247e+01
std      1.237403e+01
min      0.000000e+00
25%      8.000000e+00
50%      1.130000e+01
75%      1.680000e+01
max      6.266700e+02
Name: total_amount, dtype: float64

time: 828 ms
```

In [110]:

```
print(df['total_amount'].describe()[7])
print(df['total_amount'].describe()[6])
print(df['total_amount'].describe()[5])
print(df['total_amount'].describe()[4])
```

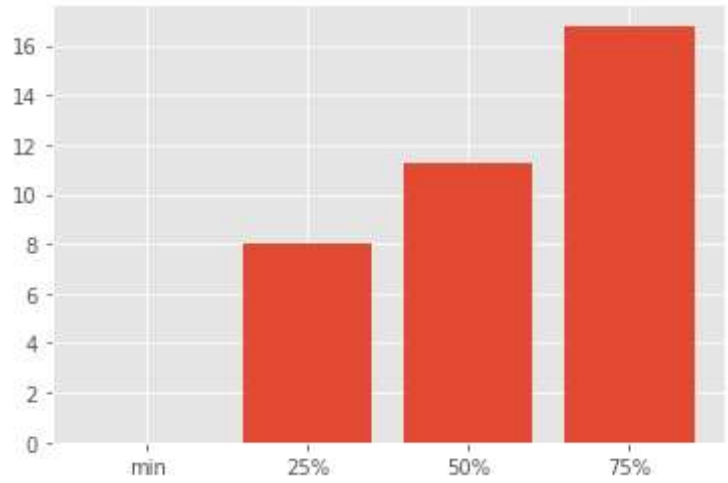
```
626.67
16.8
11.3
8.0
time: 3.24 s
```

In [111]:

```
plt.bar(df['total_amount'].describe().keys()[3:7], df['total_amount'].describe()[3:7])
```

Out[111]:

```
<BarContainer object of 4 artists>
```




```
time: 2.08 s
```

💡 ***The typical amount for taxi trips in New York Mar-May 2014 has been $11.30.***

In [151]:

```
orders_by_total_amount_dict = {'high amount ($50+)':         100 * sum( (df['total_amount']
orders_by_total_amount_dict['more thant typical ($20+)']  = 100 * sum( (df['total_amount']
orders_by_total_amount_dict['typical ($10-$20)']          = 100 * sum( (df['total_amount']
orders_by_total_amount_dict['low amount (less than $10)'] = 100 * sum( (df['total_amount']
orders_by_total_amount_dict['void']                       = 100 * sum( (df['total_amount']
orders_by_total_amount_dict
```

Out[151]:

```
{'high amount ($50+)': 3.082307857014979,
 'more thant typical ($20+)': 14.298152942405702,
 'typical ($10-$20)': 40.39838267784813,
 'low amount (less than $10)': 42.221008131739076,
 'void': 0.0001483909921081306}
```

time: 10.3 s

In [152]:

```
sum(orders_by_total_amount_dict.values())
```

Out[152]:

```
100.0
```

time: 0 ns

In [153]:

```
orders_by_total_amount = pd.DataFrame(
                      orders_by_total_amount_dict.values(),
              index = orders_by_total_amount_dict.keys(),
              columns = ['%'])
orders_by_total_amount.sort_values(by='%',ascending=False, inplace=True)
orders_by_total_amount
```

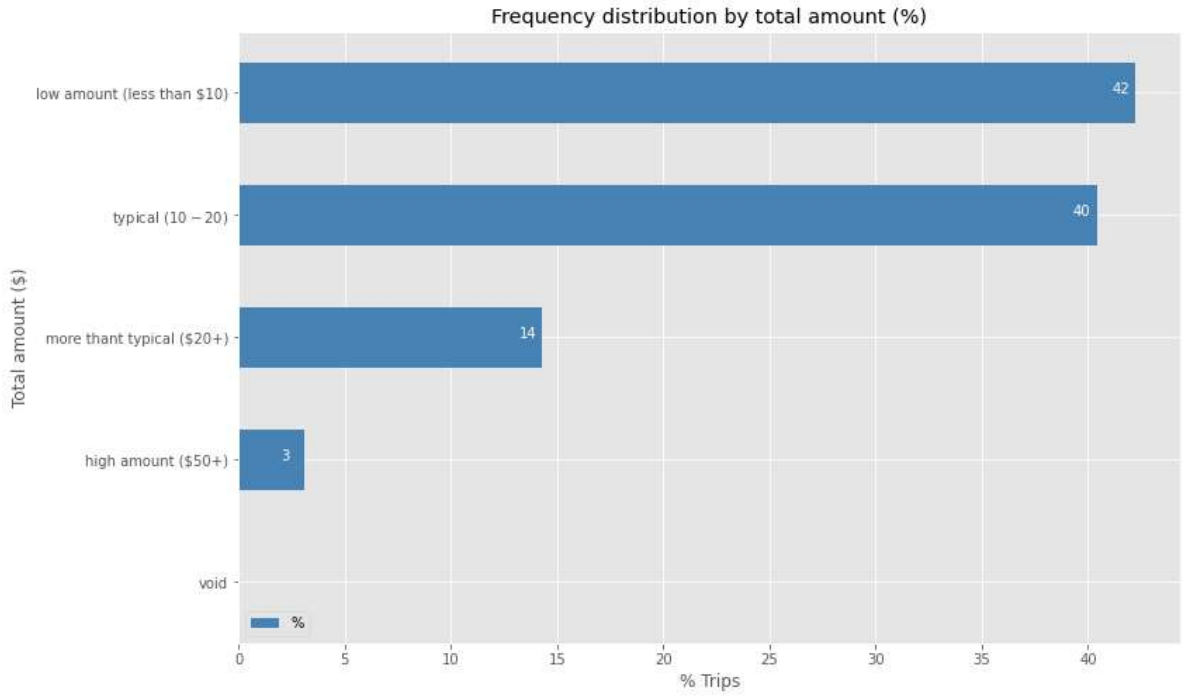
Out[153]:

	%
low amount (less than $10)	42.221008
typical ($10–$20)	40.398383
more thant typical ($20+)	14.298153
high amount ($50+)	3.082308
void	0.000148

time: 15 ms

In [155]:

```
## Distribution %

orders_by_total_amount.sort_values(by='%',ascending=True, inplace=True)
orders_by_total_amount.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Trips')
plt.ylabel('Total amount ($)')
plt.title('Frequency distribution by total amount (%)')

for i, value in enumerate(orders_by_total_amount['%']):
    label = format(int(value))

    # place text at the end of bar (subtracting 1.1% from x, and 0 from y to make it fit wi
    plt.annotate(label, xy=(value - 1.1, i - 0), color='white')

plt.show()
```



time: 359 ms

💡 ***Typical amount: $11.30.***

💡 ***82% of trips amount to up to $20.*** 💡 ***42% of them amount to less than $10 .***

Trips: Frequency distribution by average speed, mph (%)

In [156]:

```
df['mph'].describe()
```

Out[156]:

```
count    1.145622e+07
mean     1.367559e+01
std      1.304318e+02
min      0.000000e+00
25%      8.300000e+00
50%      1.127610e+01
75%      1.526250e+01
max      1.263600e+05
Name: mph, dtype: float64

time: 1.09 s
```

In [157]:

```
print(df['mph'].describe()[7])
print(df['mph'].describe()[6])
print(df['mph'].describe()[5])
print(df['mph'].describe()[4])
```

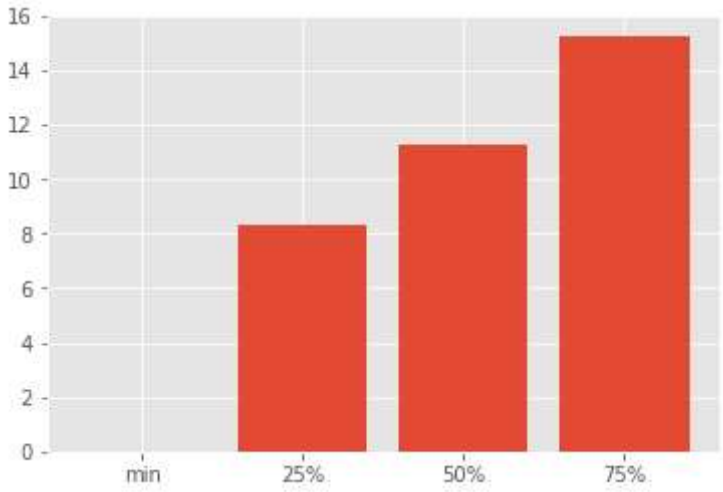
```
126360.0
15.2625
11.276102088167054
8.3
time: 3.22 s
```

In [158]:

```
plt.bar(df['mph'].describe().keys()[3:7], df['mph'].describe()[3:7])
```

Out[158]:

```
<BarContainer object of 4 artists>
```




```
time: 1.77 s
```

The typical speed of taxi trips in New York Mar-May 2014 has been 11.3 mph.

In [167]:

```
orders_by_mph_dict = {'high speed (30+ mph)':           100 * sum( (df['mph'] <= 126360) &
orders_by_mph_dict['above typical speed (15-30 mph)'] = 100 * sum( (df['mph'] <=     30) &
orders_by_mph_dict['typical (low) speed (10-15 mph)'] = 100 * sum( (df['mph'] <=     15) &
orders_by_mph_dict['very low speed (below 10 mph)']   = 100 * sum( (df['mph'] <=     10) &
orders_by_mph_dict['void']                            = 100 * sum( (df['mph'] ==      0)
orders_by_mph_dict
```

Out[167]:

```
{'high speed (30+ mph)': 2.6470247038704997,
 'above typical speed (15-30 mph)': 23.393508208335017,
 'typical (low) speed (10-15 mph)': 34.52408957543679,
 'very low speed (below 10 mph)': 38.82141414695125,
 'void': 0.613963365406446}
```

time: 10.9 s

In [168]:

```
sum(orders_by_mph_dict.values())
```

Out[168]:

```
100.0
```

time: 15 ms

In [169]:

```
orders_by_mph = pd.DataFrame(
                       orders_by_mph_dict.values(),
              index = orders_by_mph_dict.keys(),
              columns = ['%'])
orders_by_mph.sort_values(by='%',ascending=False, inplace=True)
orders_by_mph
```

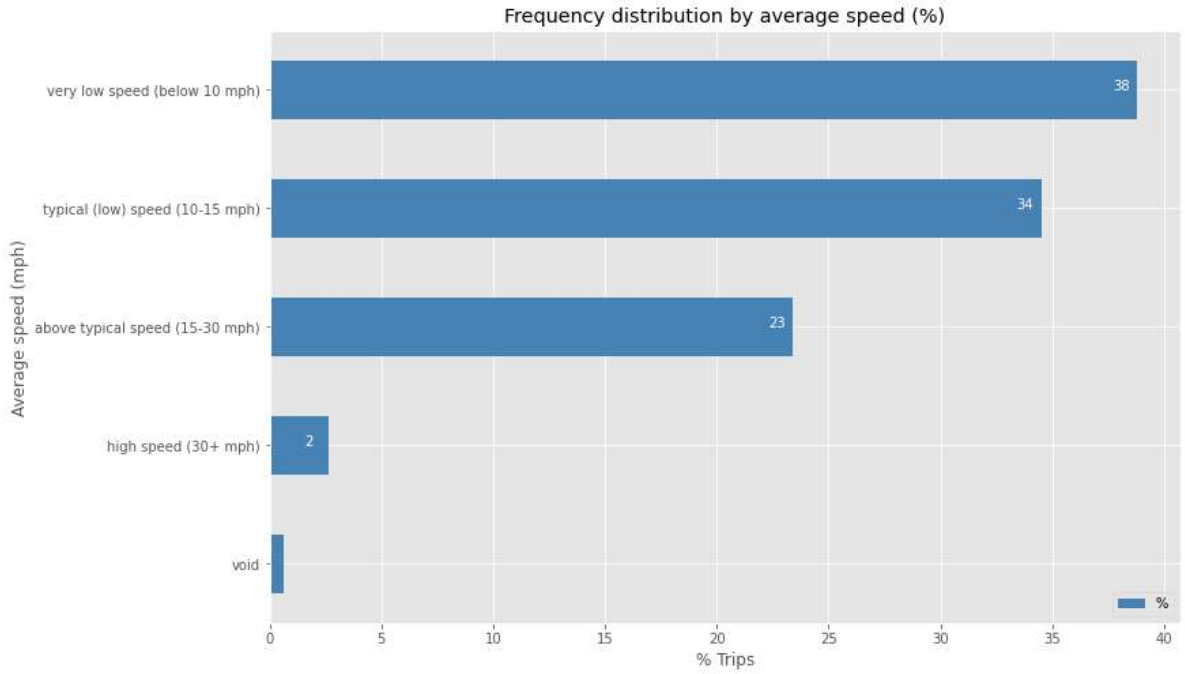
Out[169]:

	%
very low speed (below 10 mph)	38.821414
typical (low) speed (10-15 mph)	34.524090
above typical speed (15-30 mph)	23.393508
high speed (30+ mph)	2.647025
void	0.613963

time: 125 ms

In [174]:

```
## Distribution %

orders_by_mph.sort_values(by='%',ascending=True, inplace=True)
orders_by_mph.plot(kind='barh', figsize=(12, 8), color='steelblue')
plt.xlabel('% Trips')
plt.ylabel('Average speed (mph)')
plt.title('Frequency distribution by average speed (%)')

for i, value in enumerate(orders_by_mph['%']):
    label = format(int(value))

    # place text at the end of bar (subtracting 1.1% from x, and 0 from y to make it fit wi
    plt.annotate(label, xy=(value - 1.1, i - 0), color='white')


plt.savefig('./img/tst.jpg')
plt.show()
```



time: 640 ms

💡 ***Typical speed: 11.3 mph.***

💡 ***95% of trips average at up to 30 mph.*** 💡 ***72% of them at less than 15 mph .***

Fine! ✔ ✔ ✔

Exporting results for retrieval

In [38]:

```
df.to_csv('df_EDA.csv')
```

```
time: 8min 45s
```

*Note: it seems, we could have rich vizes pretty efficently using the « **lux** Python API for intelligent visual data discovery» (abc.html) library.*
Well, it's still to work here (getting advice from community).

Next: Modeling: Time Series Analysis • Forecasting (99__Time_Series_Analysis_Forecasting.ipynb)