Prediction of the accident possibility for carsharing company

Content

Project Description

Carsharing company would like to develop system for the evaluation of the risk of accident for the selected route. As risk company understands is possibility of accident with any damage to the vehicle. The system has to evaluate the risk level just after the booking of the car by client. Current task for the company is to understand is it possible to predict the possibility of accident based on the historical data of one of the regions where company operated.

Main tasks are following:

1) To connect to database and import data;

2) Prepare and analyze the data.

3) Set the tasks to work team.

4) Train the models and select the best one;

5) Find the key factor leading to possibility of accident;

6) Propose the tools for reduction of possibility of accident.

Connection to database and data loading

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sqlalchemy import create_engine
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from sklearn.preprocessing import OneHotEncoder,StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import precision_recall_curve
```

In [2]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=RuntimeWarning)
```

In [3]:
```
db_config = {
'user': '******', # имя пользователя,
'pwd': '*****', # пароль,
'host': '****',
'port': ****, # порт подключения,
'db': '****' # название базы данных,
}
```

In [4]:
```
connection_string = 'postgresql://{}:{}@{}:{}/{}'.format(
    db_config['user'],
```

```python
    db_config['pwd'],
    db_config['host'],
    db_config['port'],
    db_config['db'],
)
```

In [5]:
```python
engine = create_engine(connection_string)
```

In [6]:
```python
query = '''
SELECT *
FROM Parties
'''

parties_df = pd.read_sql_query(query, con=engine)
```

In [7]:
```python
query = '''
SELECT *
FROM collisions
'''

collisions_df = pd.read_sql_query(query, con=engine)
```

In [8]:
```python
query = '''
SELECT *
FROM Vehicles
'''

vehicles_df = pd.read_sql_query(query, con=engine)
```

In [9]:
```python
query = '''
SELECT *
FROM case_ids
'''

case_ids_df = pd.read_sql_query(query, con=engine)
```

Data overview

In [10]:
```
parties_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2752408 entries, 0 to 2752407
Data columns (total 9 columns):
 #   Column               Dtype  
---  ------               -----  
 0   id                   int64  
 1   case_id              object 
 2   party_number         int64  
 3   party_type           object 
 4   at_fault             int64  
 5   insurance_premium    float64
 6   party_sobriety       object 
 7   party_drug_physical  object 
 8   cellphone_in_use     float64
dtypes: float64(2), int64(3), object(4)
memory usage: 189.0+ MB
```

In [11]:
```
parties_df.head()
```

Out[11]:

	id	case_id	party_number	party_type	at_fault	insurance_premium	party_sobriety	party_drug_physical	cellphone_in_use
0	22	3899454	1	road signs	1	29.0	had not been drinking	None	0.0
1	23	3899454	2	road signs	0	7.0	had not been drinking	None	0.0
2	29	3899462	2	car	0	21.0	had not been drinking	None	0.0
3	31	3899465	2	road signs	0	24.0	had not been drinking	None	0.0
4	41	3899478	2	road bumper	0	NaN	not applicable	not applicable	0.0

In [12]:
```
collisions_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1400000 entries, 0 to 1399999
Data columns (total 20 columns):
 #   Column                      Non-Null Count    Dtype  
---  ------                      --------------    -----  
 0   case_id                     1400000 non-null  object 
 1   county_city_location        1400000 non-null  object 
 2   county_location             1400000 non-null  object 
 3   distance                    1400000 non-null  float64
 4   direction                   1059358 non-null  object 
 5   intersection                1387781 non-null  float64
 6   weather_1                   1392741 non-null  object 
 7   location_type               518779 non-null   object 
 8   collision_damage            1400000 non-null  object 
 9   party_count                 1400000 non-null  int64  
 10  primary_collision_factor    1391834 non-null  object 
 11  pcf_violation_category      1372046 non-null  object 
 12  type_of_collision           1388176 non-null  object 
 13  motor_vehicle_involved_with  1393181 non-null  object 
 14  road_surface                1386907 non-null  object 
 15  road_condition_1            1388012 non-null  object 
 16  lighting                    1391407 non-null  object 
 17  control_device              1391593 non-null  object 
 18  collision_date              1400000 non-null  object 
 19  collision_time              1387692 non-null  object 
dtypes: float64(2), int64(1), object(17)
memory usage: 213.6+ MB
```

In [13]:
```
collisions_df.head()
```

Out[13]:

	case_id	county_city_location	county_location	distance	direction	intersection	weather_1	location_type	collision_damage	party_count	primary_coll
0	4083072	1942	los angeles	528.0	north	0.0	cloudy	highway	small damage	2	vehicle c
1	4083075	4313	santa clara	0.0	None	1.0	clear	None	small damage	1	vehicle c
2	4083073	0109	alameda	0.0	None	1.0	clear	None	scratch	2	vehicle c
3	4083077	0109	alameda	0.0	None	1.0	clear	None	scratch	2	vehicle c
4	4083087	4313	santa clara	0.0	None	1.0	clear	None	scratch	2	vehicle c

In [14]:
```
vehicles_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1021234 entries, 0 to 1021233
Data columns (total 6 columns):
 #   Column                Non-Null Count    Dtype  
---  ------                --------------    -----  
 0   id                    1021234 non-null  int64  
 1   case_id               1021234 non-null  object 
 2   party_number          1021234 non-null  int64  
 3   vehicle_type          1021234 non-null  object 
 4   vehicle_transmission  997575 non-null   object 
 5   vehicle_age           996652 non-null   float64
dtypes: float64(1), int64(2), object(3)
memory usage: 46.7+ MB
```

In [15]:
```
vehicles_df.head()
```

Out[15]:

	id	case_id	party_number	vehicle_type	vehicle_transmission	vehicle_age
0	1175713	5305032	2	sedan	manual	3.0
1	1	3858022	1	sedan	auto	3.0
2	1175712	5305030	1	sedan	auto	3.0
3	1175717	5305033	3	sedan	auto	5.0
4	1175722	5305034	2	sedan	auto	5.0

In [16]:

```
del vehicles_df
del parties_df
del collisions_df
```

Conclusion

1) Parties_df has 2 752 408 rows and 9 columns;

2) collisions_df has 1 400 000 rows and 6 columns;

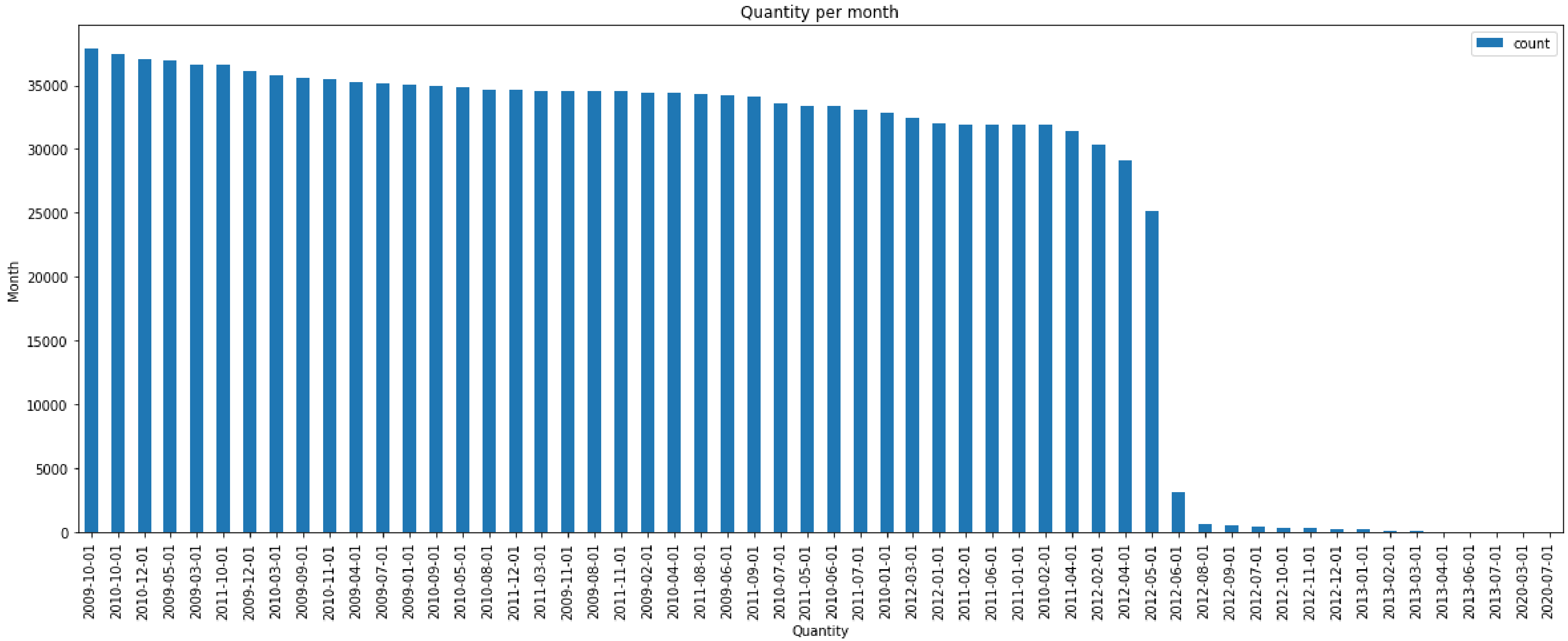
3) vehicles_df has 1 021 234 rows and 5 columns;

Statistical analysis of accident factors

In [17]:

```
# dispaly the quantity of accidents per month
query = '''
SELECT COUNT(case_id), DATE(DATE_TRUNC('month',collision_date)) AS month_date
FROM collisions
GROUP BY DATE(DATE_TRUNC('month',collision_date))
ORDER BY COUNT(case_id) DESC
'''

month_df = pd.read_sql_query(query, con=engine)
month_df.head()
month_df.plot(kind='bar',y = 'count', x = 'month_date', figsize =(20,7),title = 'Quantity per month')
plt.xlabel('Quantity')
```

```
plt.ylabel('Month')
plt.show()
```



Tasks to work team:

1) To analyze the level of damage to vehicles, based on the surface condition in the moment of accident (to join collisions и parties)

2) Find the most often reasons of accidents (table parties)

3) Analyze in what period of the day is the highest quantity of accidents with fatal damage happens.

4) Does the type of gearboxe affect on the quantity of accidents .

5) Analysis of relation of level of damage and drivers sbriety.

6) Which region has and in whic month has the highest quantity of accdents.

the numbers 4 and 5 to be analysed below

Analysis of relation of level of damage and drivers sbriety

In [18]:
```
# query to get the statistic on not sober drivers
query = '''
SELECT c.collision_damage,COUNT(c.case_id)
  FROM collisions AS c
  INNER JOIN Parties AS p
       ON c.case_id = p.case_id
  WHERE p.party_sobriety LIKE '%%had been drinking%%'
 GROUP BY c.collision_damage
 ORDER BY COUNT(c.case_id) DESC
'''

dmg_drunk_df = pd.read_sql_query(query, con=engine)
dmg_drunk_df['percentage'] = round(dmg_drunk_df['count'] / dmg_drunk_df['count'].sum()*100,0)
dmg_drunk_df
```

Out[18]:

	collision_damage	count	percentage
0	small damage	87228	58.0
1	middle damage	27876	18.0
2	scratch	23937	16.0
3	severe damage	7956	5.0
4	fatal	4485	3.0

In [20]:
```
# query to get the statistic on sober drivers
query = '''
SELECT c.collision_damage,COUNT(c.case_id)
  FROM collisions AS c
  FULL OUTER JOIN Parties AS p
       ON c.case_id = p.case_id
  WHERE p.party_sobriety NOT LIKE '%%had not been drinking%%'
 GROUP BY c.collision_damage
 ORDER BY COUNT(c.case_id) DESC
'''

dmg_sober_df = pd.read_sql_query(query, con=engine)
```

```
dmg_sober_df['percentage'] = round(dmg_sober_df['count'] / dmg_sober_df['count'].sum()*100,0)
dmg_sober_df
```

Out[20]:

	collision_damage	count	percentage
0	small damage	394814	71.0
1	scratch	79614	14.0
2	middle damage	57497	10.0
3	severe damage	14771	3.0
4	fatal	7030	1.0

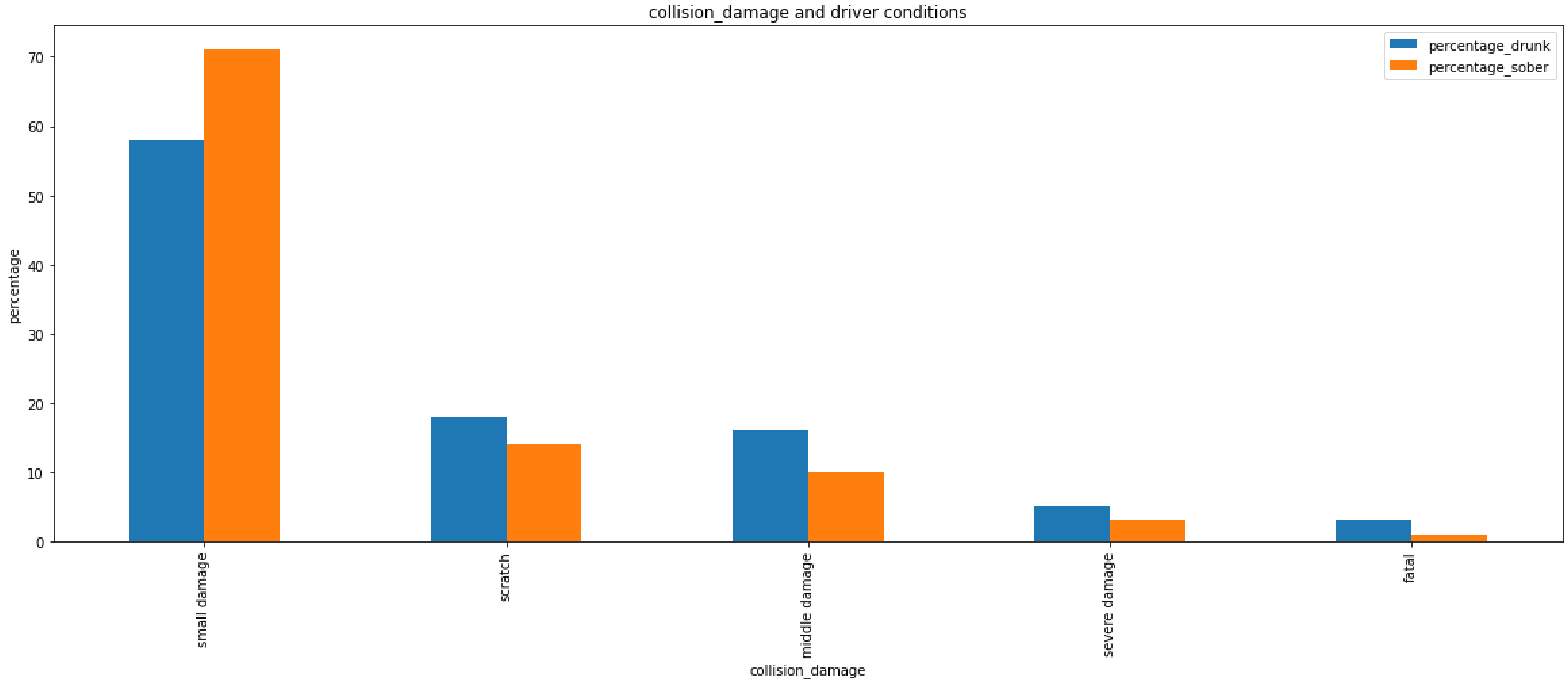
In [21]:

```
# Comparison of sober and drunk driver statistic
sober_drunk = pd.DataFrame(dmg_sober_df['collision_damage'])
sober_drunk['percentage_drunk'] = dmg_drunk_df['percentage']
sober_drunk['percentage_sober'] = dmg_sober_df['percentage']
print(sober_drunk)
sober_drunk.plot.bar(x='collision_damage', figsize =(20,7),title = 'collision_damage and driver conditions')
plt.xlabel('collision_damage')
plt.ylabel('percentage')
```

```
  collision_damage  percentage_drunk  percentage_sober
0     small damage              58.0              71.0
1          scratch              18.0              14.0
2    middle damage              16.0              10.0
3    severe damage               5.0               3.0
4            fatal               3.0               1.0
```

Out[21]:

```
Text(0, 0.5, 'percentage')
```



In [22]:
```
sober_drunk['percentage_sober_drunk'] = round(( sober_drunk['percentage_drunk']/sober_drunk['percentage_sober']-1)*100,0)
```

In [23]:
```
sober_drunk
```

Out[23]:

	collision_damage	percentage_drunk	percentage_sober	percentage_sober_drunk
0	small damage	58.0	71.0	-18.0
1	scratch	18.0	14.0	29.0
2	middle damage	16.0	10.0	60.0
3	severe damage	5.0	3.0	67.0
4	fatal	3.0	1.0	200.0

Conclusion

- ДТП с небольшим уроном случаются на 18% реже в пьяном состоянии
- ДТП с царапинами случаются на 29% чаще в пьяном состоянии
- ДТП со средним уроном случаются на 60% чаще в пьяном состоянии
- ДТП с серьезным уроном случаются на 67% чаще в пьяном состоянии
- ДТП с фатальным уроном случаются на 200% чаще в пьяном состоянии

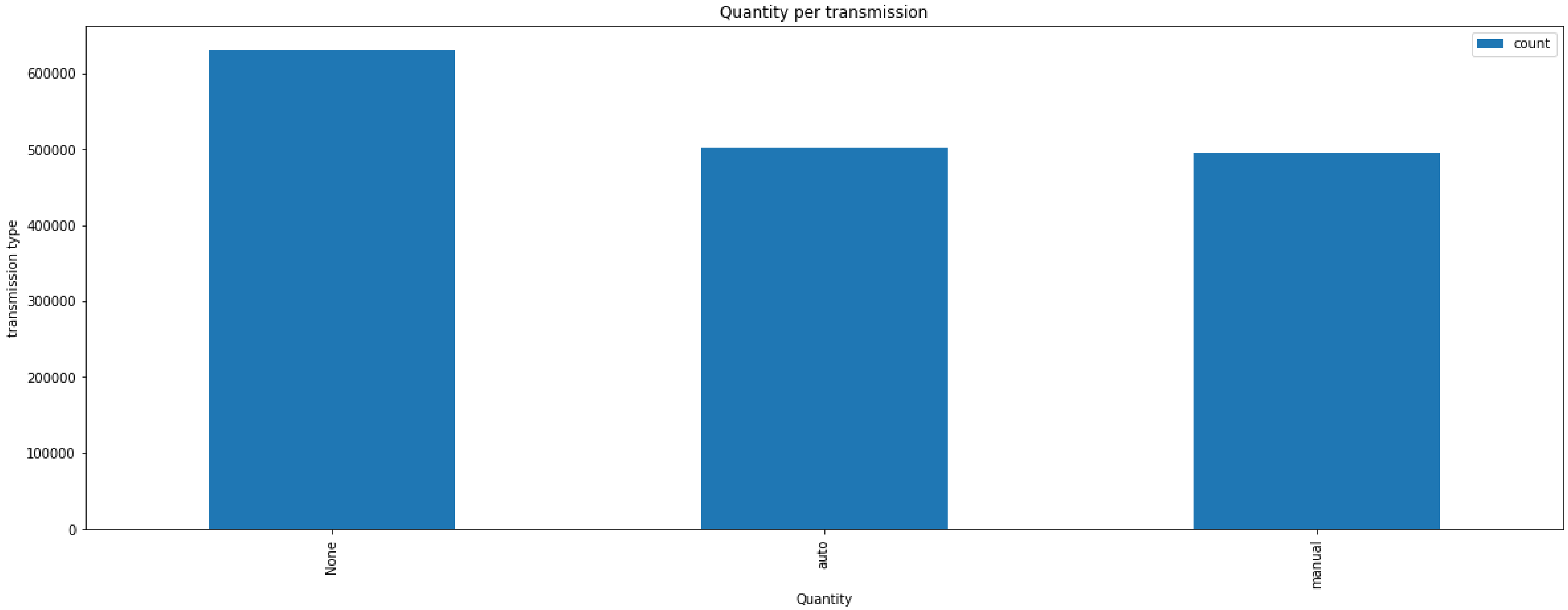
В целом по информации из статистики можно сказать что ДТП с серьезными и фатальными поврждениями случаются в разы чаще в пьяном состоянии чем в трезвом, хотя общее количество ДТП в трезвом состоянии по абсолютным цифрам больше.

Главный вывод - не пей за рулем!

Лучше закажи такси

Analysis of relation of gearbox type on quantity of accidents.

In [24]:
```
query = '''
SELECT COUNT(s.case_id), v.vehicle_transmission AS transmission_type
  FROM collisions AS s
       FULL OUTER JOIN Vehicles AS v
       ON s.case_id = v.case_id
GROUP BY v.vehicle_transmission
ORDER BY COUNT(s.case_id) DESC
'''

kpp_df = pd.read_sql_query(query, con=engine)
kpp_df.head()
kpp_df.plot(kind='bar',y = 'count', x = 'transmission_type', figsize =(20,7),title = 'Quantity per transmission')
plt.xlabel('Quantity')
plt.ylabel('transmission type')
plt.show()
```



In [25]:
```
del kpp_df
del dmg_sober_df
```

Conclusion

Based on the analysis the conclusion is following - type of gearbox has no affect on quantity of accidents.

Creating a table with all features for statistic analysis

In [26]:
```
query = '''
SELECT *
  FROM collisions AS s
       INNER JOIN Vehicles AS v
       ON s.case_id = v.case_id
       INNER JOIN Parties AS p
       ON s.case_id = p.case_id
'''
```

```python
total_df = pd.read_sql_query(query, con=engine)
total_df.head()
```

Out[26]:

	case_id	county_city_location	county_location	distance	direction	intersection	weather_1	location_type	collision_damage	party_count	...	vehicle_a
0	4014984	3631	san bernardino	0.0	None	1.0	clear	highway	fatal	2	...	1
1	4027576	3711	san diego	350.0	south	0.0	clear	None	middle damage	1	...	
2	4033928	2002	madera	55.0	west	0.0	cloudy	highway	small damage	2	...	
3	4035554	1005	fresno	0.0	None	1.0	clear	ramp	scratch	2	...	
4	4035554	1005	fresno	0.0	None	1.0	clear	ramp	scratch	2	...	

5 rows × 35 columns

In [27]:

```python
total_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2194043 entries, 0 to 2194042
Data columns (total 35 columns):
 #   Column                       Dtype  
---  ------                       -----  
 0   case_id                      object 
 1   county_city_location         object 
 2   county_location              object 
 3   distance                     float64
 4   direction                    object 
 5   intersection                 float64
 6   weather_1                    object 
 7   location_type                object 
 8   collision_damage             object 
 9   party_count                  int64  
 10  primary_collision_factor     object 
 11  pcf_violation_category       object 
 12  type_of_collision            object 
 13  motor_vehicle_involved_with  object 
 14  road_surface                 object 
 15  road_condition_1             object 
 16  lighting                     object 
 17  control_device               object 
 18  collision_date               object 
 19  collision_time               object 
 20  id                           int64  
 21  case_id                      object 
 22  party_number                 int64  
 23  vehicle_type                 object 
 24  vehicle_transmission         object 
 25  vehicle_age                  float64
 26  id                           int64  
 27  case_id                      object 
 28  party_number                 int64  
 29  party_type                   object 
 30  at_fault                     int64  
 31  insurance_premium            float64
 32  party_sobriety               object 
 33  party_drug_physical          object 
 34  cellphone_in_use             float64
dtypes: float64(5), int64(6), object(24)
memory usage: 585.9+ MB
```

In [28]:
```python
# deletion of useless columns
total_df = total_df.drop(columns = ['id','case_id'])
total_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2194043 entries, 0 to 2194042
Data columns (total 30 columns):
 #   Column                       Dtype  
---  ------                       -----  
 0   county_city_location         object 
 1   county_location              object 
 2   distance                     float64
 3   direction                    object 
 4   intersection                 float64
 5   weather_1                    object 
 6   location_type                object 
 7   collision_damage             object 
 8   party_count                  int64  
 9   primary_collision_factor     object 
 10  pcf_violation_category       object 
 11  type_of_collision            object 
 12  motor_vehicle_involved_with  object 
 13  road_surface                 object 
 14  road_condition_1             object 
 15  lighting                     object 
 16  control_device               object 
 17  collision_date               object 
 18  collision_time               object 
 19  party_number                 int64  
 20  vehicle_type                 object 
 21  vehicle_transmission         object 
 22  vehicle_age                  float64
 23  party_number                 int64  
 24  party_type                   object 
 25  at_fault                     int64  
 26  insurance_premium            float64
 27  party_sobriety               object 
 28  party_drug_physical          object 
 29  cellphone_in_use             float64
dtypes: float64(5), int64(4), object(21)
memory usage: 502.2+ MB
```

In [29]:
```
# change of datatype of column county_city_location
total_df['county_city_location'] = total_df['county_city_location'].astype('float64')
```

In [30]:
```
# selection of categorical and numeric columns
numeric_col = list(total_df.select_dtypes(include=['int64', 'float64']).columns[:])
categorical_col = list(total_df.select_dtypes(include=['object']).columns[:])
categorical_col.remove('collision_date')
categorical_col.remove('collision_time')
dt_cols = ['collision_date','collision_time']
print('Numerical columns:', numeric_col,'\n')
print('Categorical columns:','\n',categorical_col,'\n')
print('Date time columns:','\n',dt_cols)
```

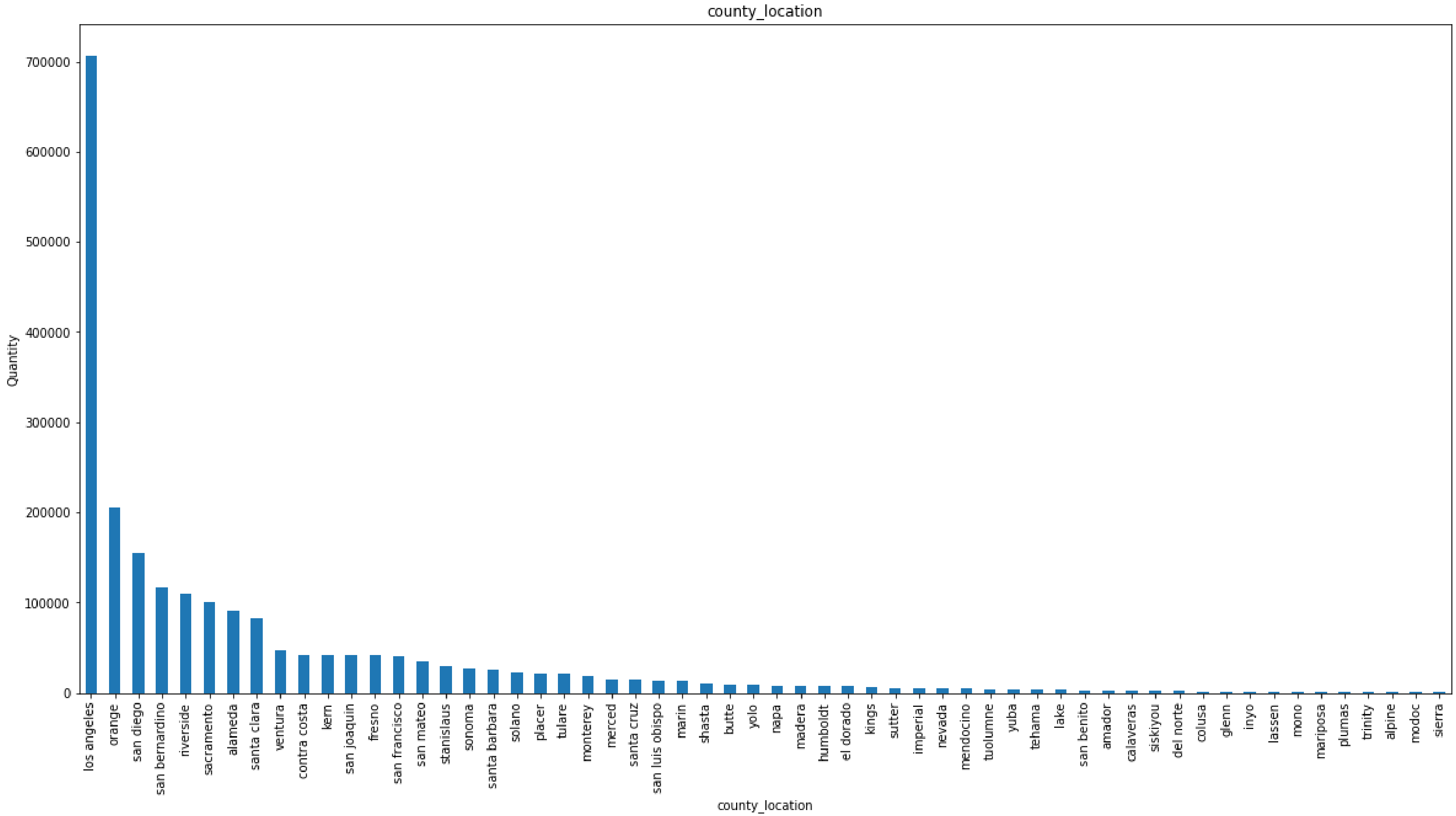
```
Numerical columns: ['county_city_location', 'distance', 'intersection', 'party_count', 'party_number', 'vehicle_age', 'party_num
ber', 'at_fault', 'insurance_premium', 'cellphone_in_use']

Categorical columns:
 ['county_location', 'direction', 'weather_1', 'location_type', 'collision_damage', 'primary_collision_factor', 'pcf_violation_c
ategory', 'type_of_collision', 'motor_vehicle_involved_with', 'road_surface', 'road_condition_1', 'lighting', 'control_device',
'vehicle_type', 'vehicle_transmission', 'party_type', 'party_sobriety', 'party_drug_physical']

Date time columns:
 ['collision_date', 'collision_time']
```

In [31]:
```
# function for plotting
def plot_bar_func (column):
    print(total_df.groupby(column)['at_fault'].count().sort_values(ascending=False))
    total_df.groupby(column)['at_fault'].count().sort_values(ascending=False).plot(kind='bar',figsize = (20,10))
    plt.title(column)
    plt.xlabel(column)
    plt.ylabel('Quantity')
    plt.show()
```

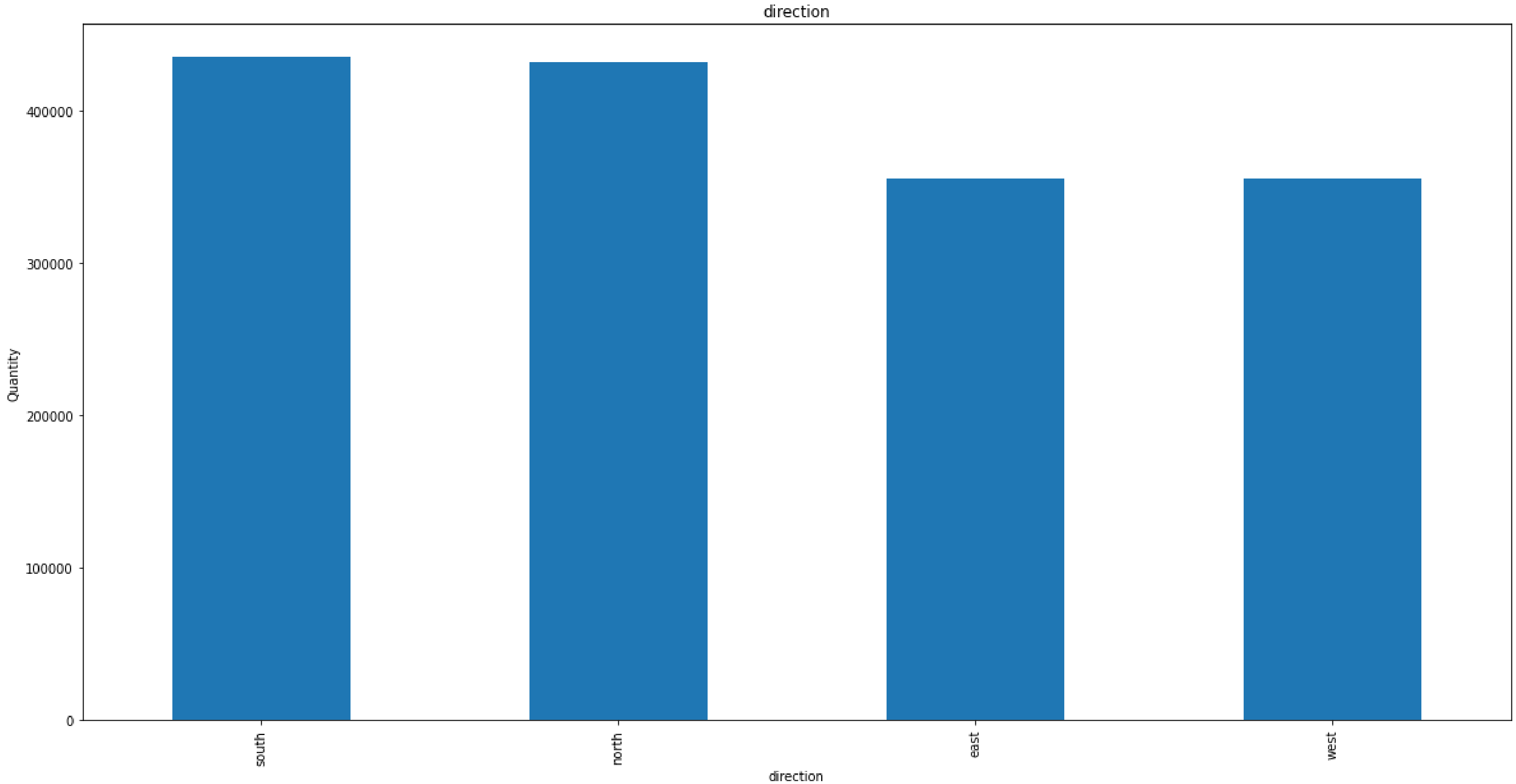
In [32]:
```
# Plotting the info of categorical columns
for i in categorical_col[:]:
    plot_bar_func(i)
```

county_location	
los angeles	706419
orange	205025
san diego	154835
san bernardino	117320
riverside	110455
sacramento	100013
alameda	90142
santa clara	82024
ventura	47227
contra costa	42282
kern	41802
san joaquin	41714
fresno	41378
san francisco	40484
san mateo	34695
stanislaus	29903
sonoma	26680
santa barbara	24939
solano	21888
placer	20961
tulare	20897
monterey	18786
merced	14607
santa cruz	14428
san luis obispo	13297
marin	13245
shasta	10728
butte	9131
yolo	8800
napa	7810
madera	7774
humboldt	7693
el dorado	7547
kings	6809
sutter	5450
imperial	5292
nevada	4987
mendocino	4305
tuolumne	3457
yuba	3254
tehama	3155
lake	2908
san benito	2585

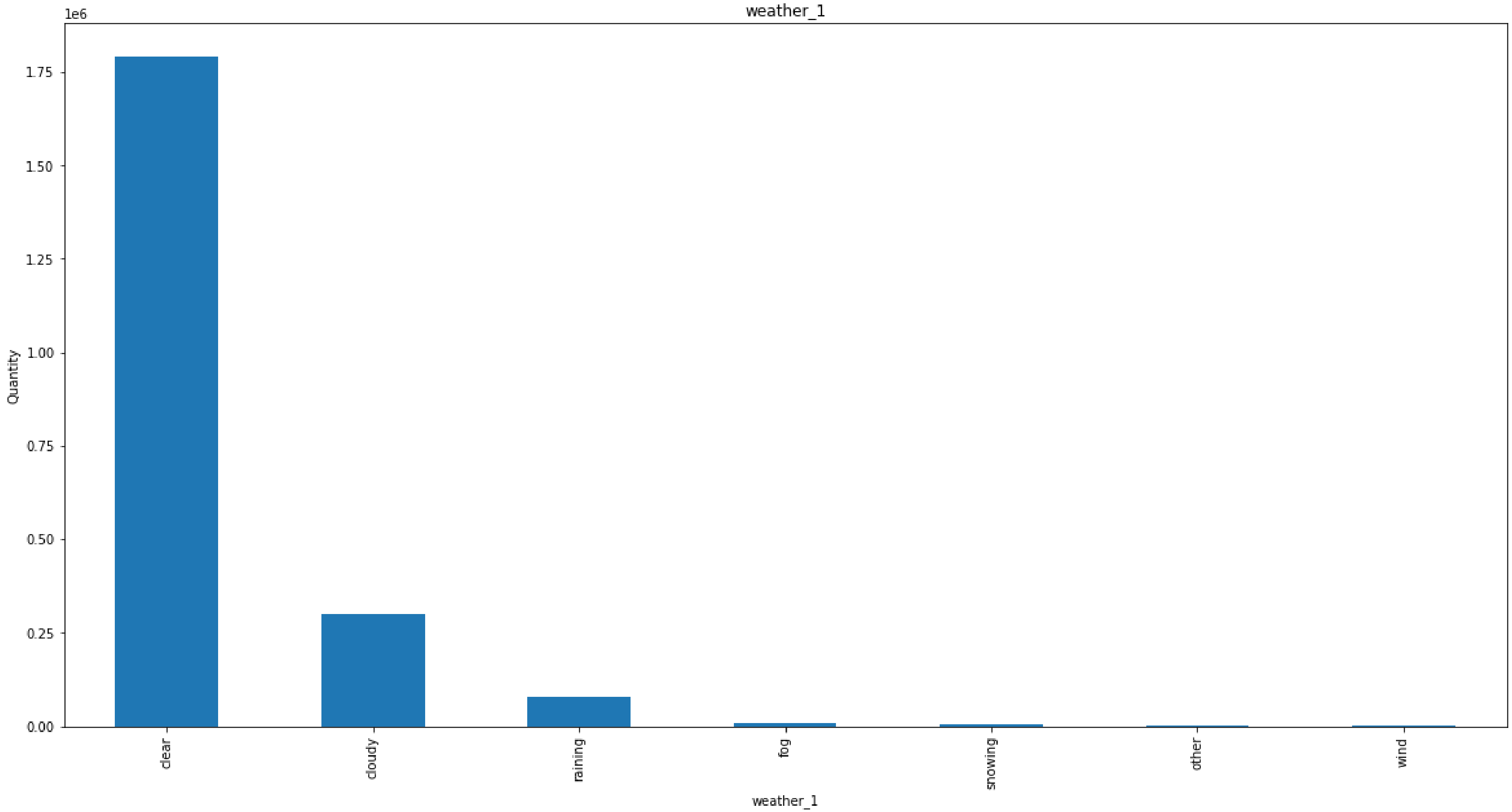
```
amador                2456
calaveras             2309
siskiyou              1831
del norte             1426
colusa                1239
glenn                 1221
inyo                  1209
lassen                1143
mono                   881
mariposa               857
plumas                 819
trinity                670
alpine                 351
modoc                  316
sierra                 184
Name: at_fault, dtype: int64
```



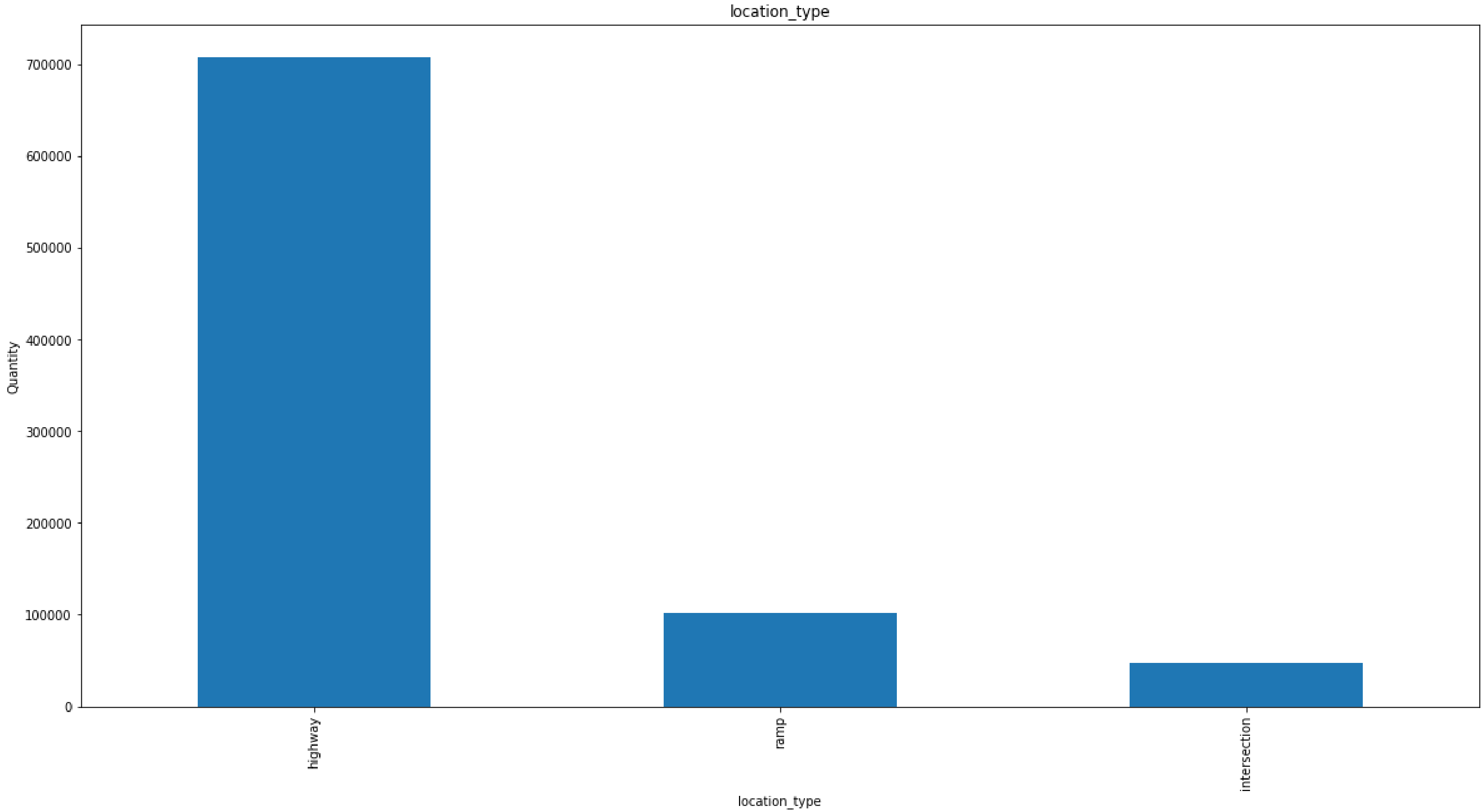

```
direction
south    435473
north    432081
east     355717
west     355433
Name: at_fault, dtype: int64
```



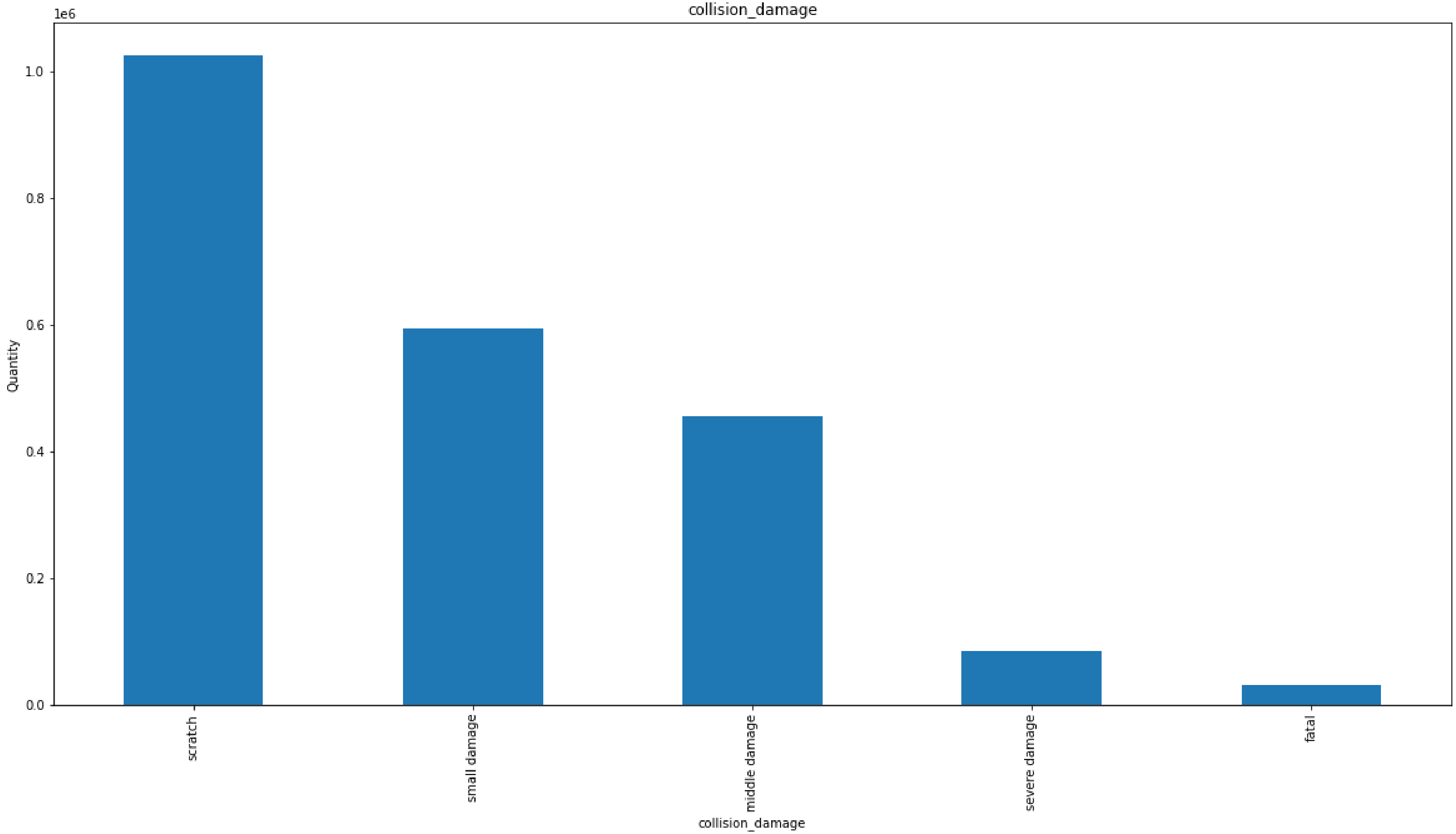

```
weather_1
clear      1792328
cloudy      300615
raining      77836
fog           7742
snowing       3624
other         1357
wind           596
Name: at_fault, dtype: int64
```



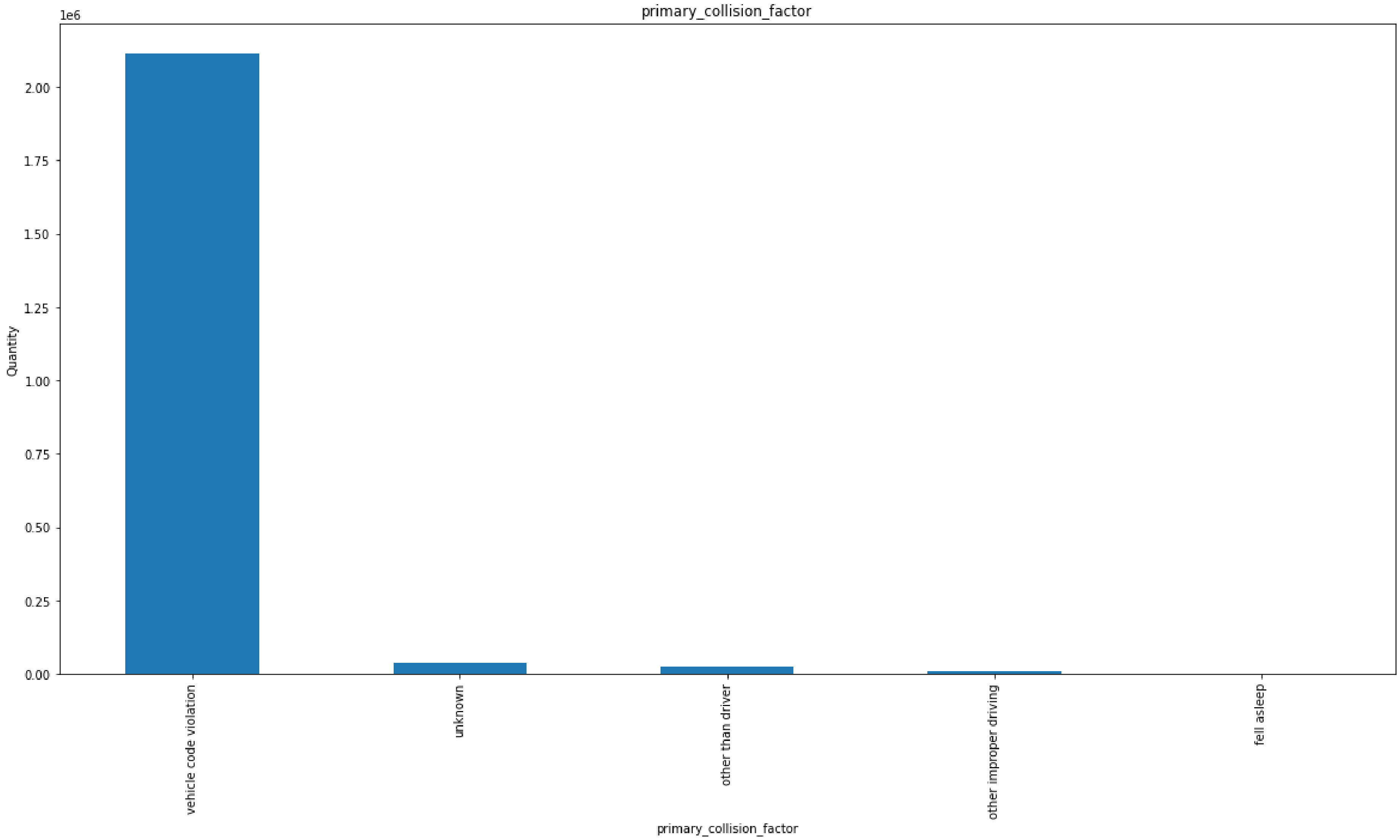

```
location_type
highway         708497
ramp            102565
intersection     47820
Name: at_fault, dtype: int64
```



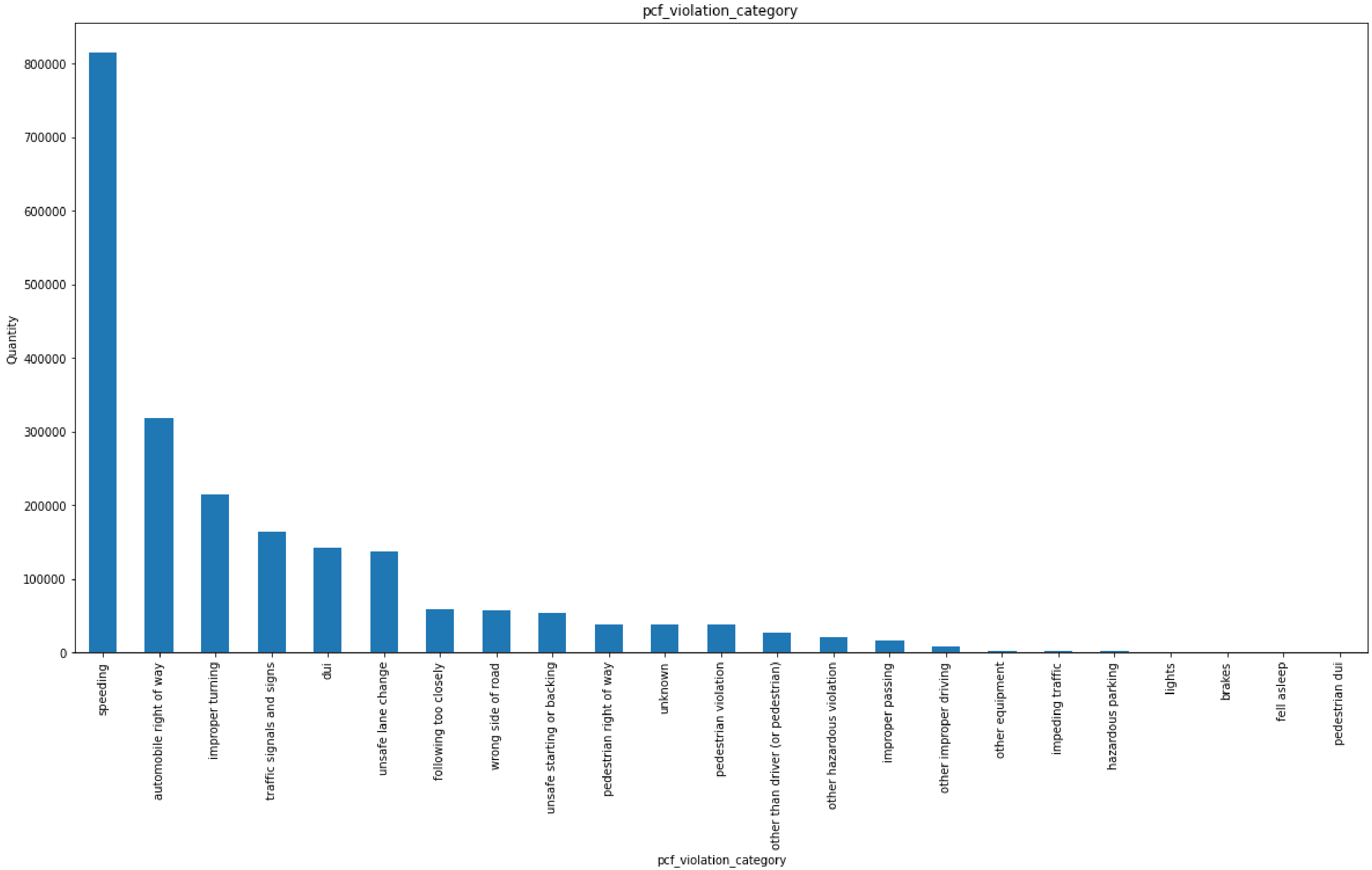

```
collision_damage
scratch          1026384
small damage      595087
middle damage     455738
severe damage      84894
fatal              31940
Name: at_fault, dtype: int64
```



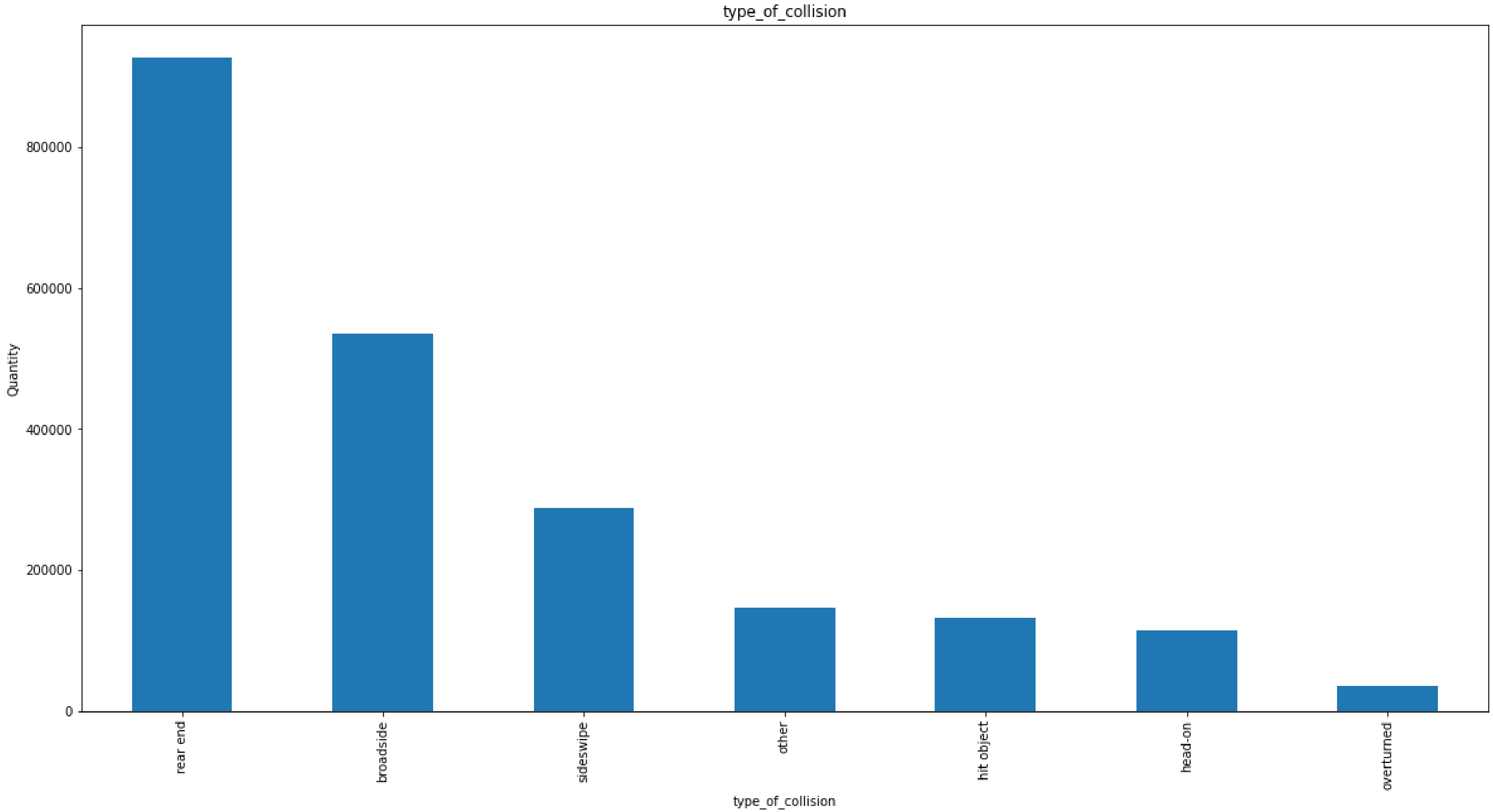

```
primary_collision_factor
vehicle code violation    2112950
unknown                     36365
other than driver           26526
other improper driving       8921
fell asleep                   111
Name: at_fault, dtype: int64
```


primary_collision_factor
1e6
Quantity
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
vehicle code violation
unknown
other than driver
other improper driving
fell asleep
primary_collision_factor

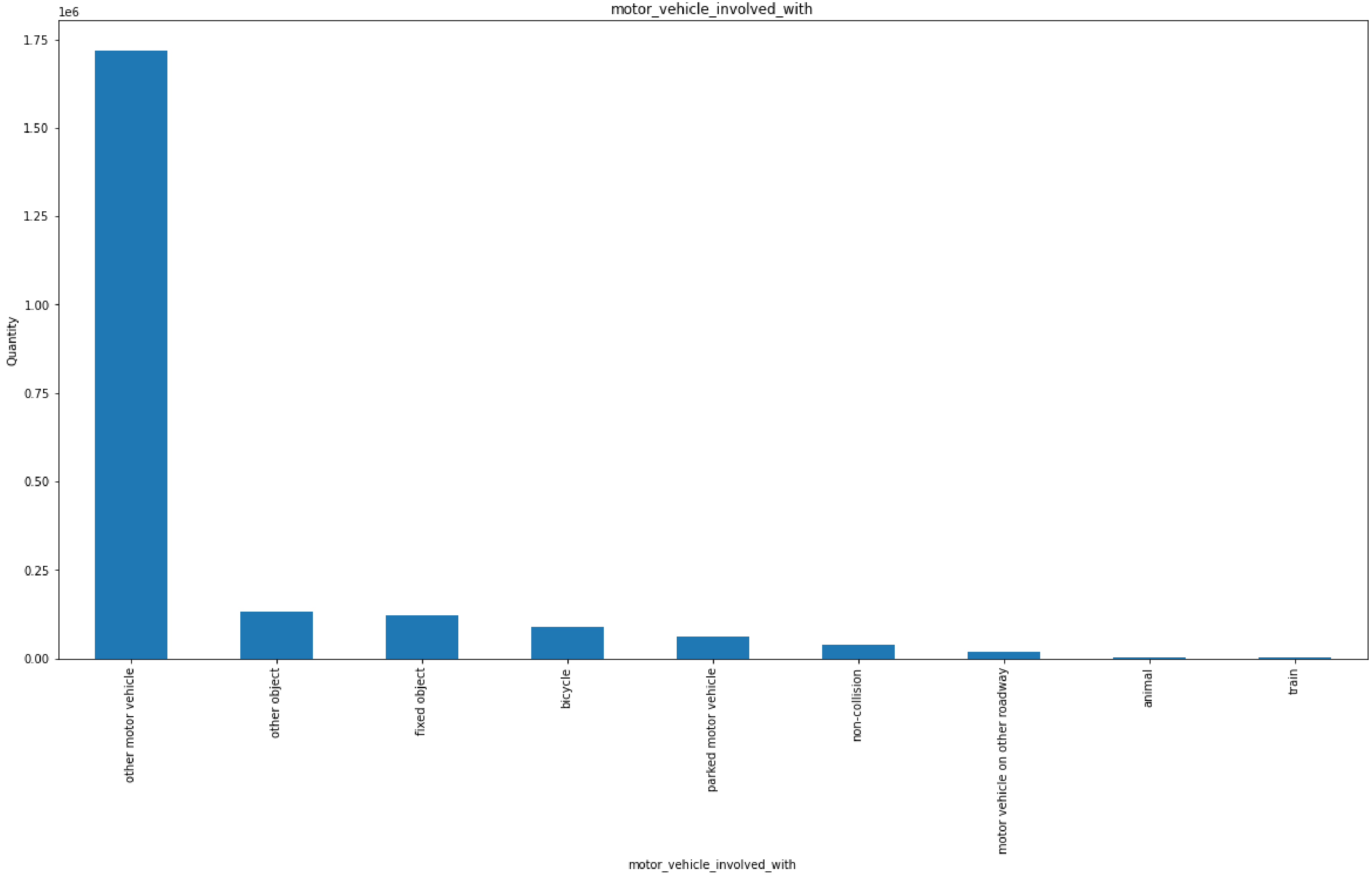
```
pcf_violation_category
speeding                             815444
automobile right of way              318257
improper turning                     215350
traffic signals and signs            164629
dui                                  142711
unsafe lane change                   137691
following too closely                 58733
wrong side of road                    57217
unsafe starting or backing            53540
pedestrian right of way               38795
unknown                               38430
pedestrian violation                  38186
other than driver (or pedestrian)     26526
other hazardous violation             21396
improper passing                      16174
other improper driving                 8921
other equipment                        1811
impeding traffic                       1694
hazardous parking                      1276
lights                                  455
brakes                                  372
fell asleep                             111
pedestrian dui                            4
Name: at_fault, dtype: int64
```


pcf_violation_category
Quantity
800000
700000
600000
500000
400000
300000
200000
100000
0
speeding
automobile right of way
improper turning
traffic signals and signs
dui
unsafe lane change
following too closely
wrong side of road
unsafe starting or backing
pedestrian right of way
unknown
pedestrian violation
other than driver (or pedestrian)
other hazardous violation
improper passing
other improper driving
other equipment
impeding traffic
hazardous parking
lights
brakes
fell asleep
pedestrian dui
pcf_violation_category

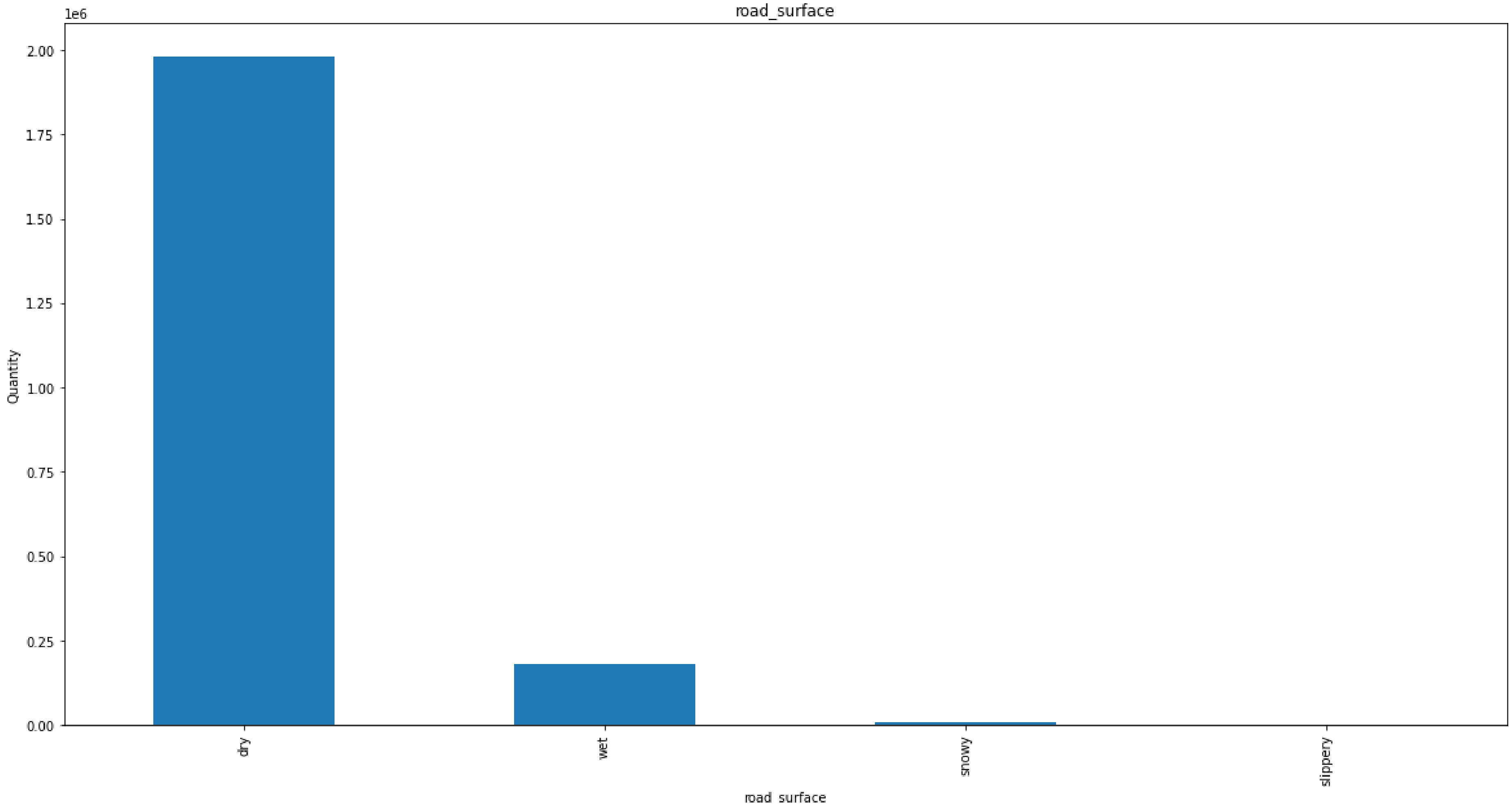
```
type_of_collision
rear end       927529
broadside      535425
sideswipe      288132
other          146999
hit object     131288
head-on        113380
overturned      35667
Name: at_fault, dtype: int64
```



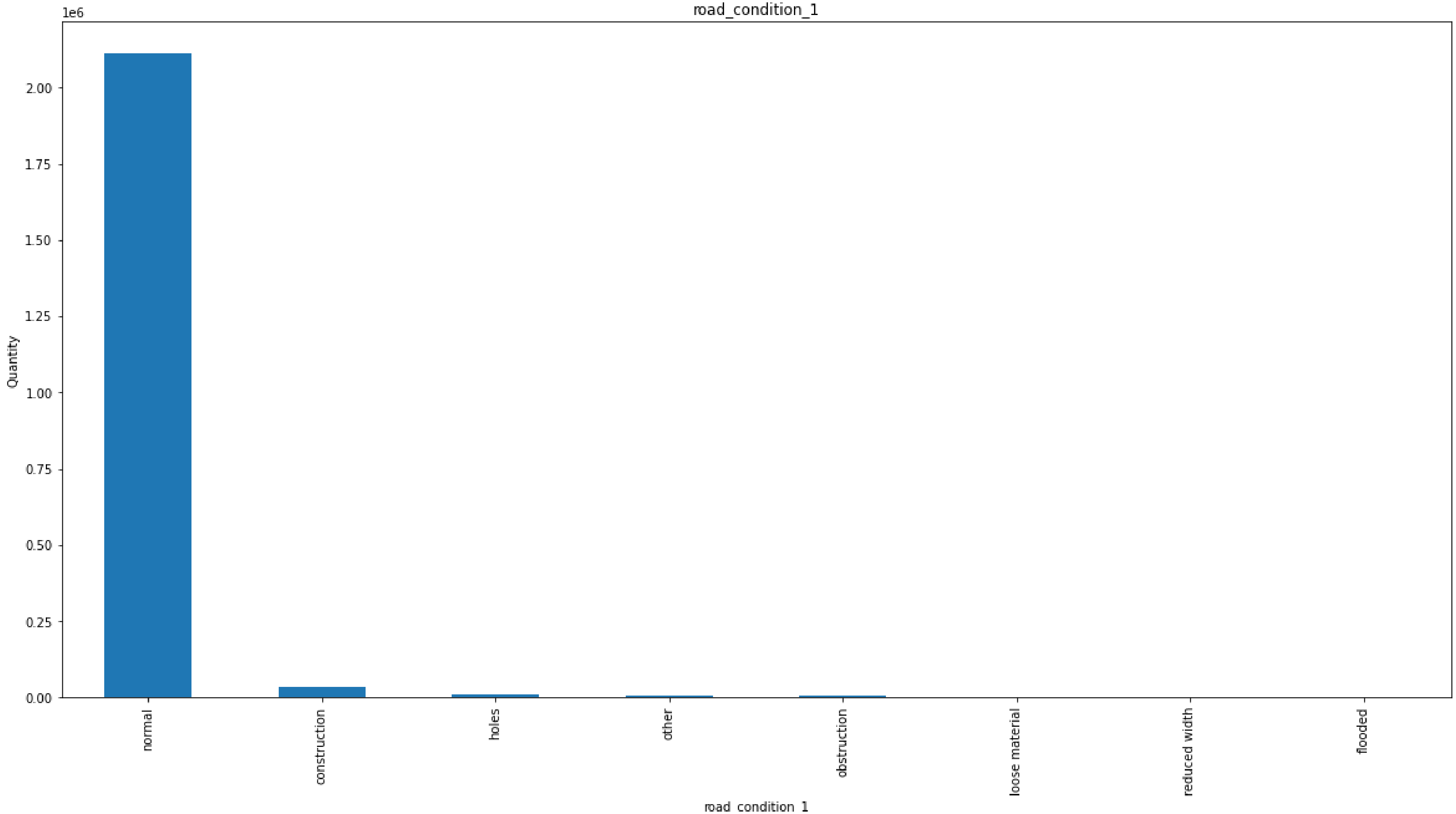
```
motor_vehicle_involved_with
other motor vehicle              1719087
other object                      131762
fixed object                      122610
bicycle                            89665
parked motor vehicle               62292
non-collision                      37920
motor vehicle on other roadway     19518
animal                              2762
train                                686
Name: at_fault, dtype: int64
```



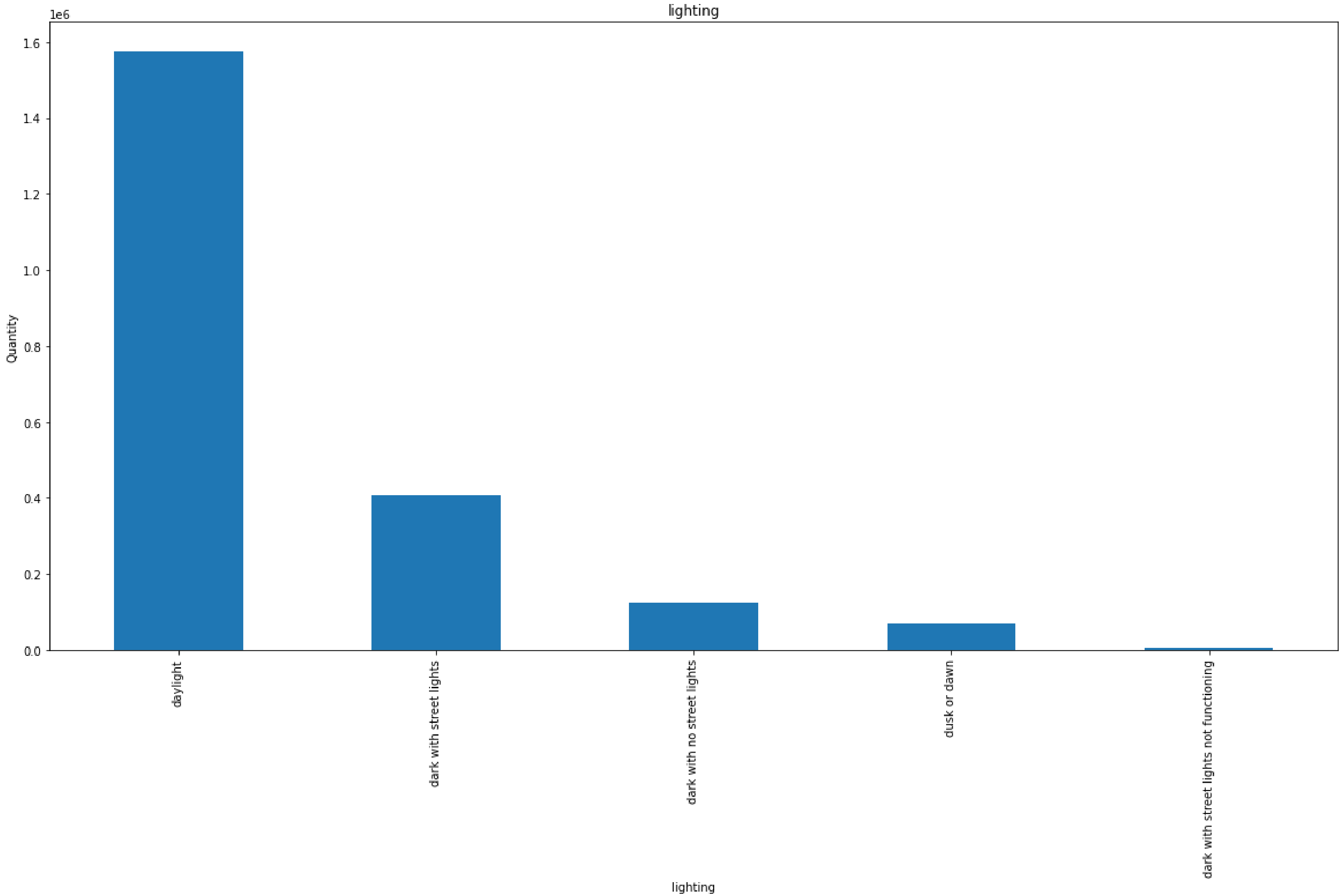

```
road_surface
dry         1983149
wet          182005
snowy          8736
slippery       1596
Name: at_fault, dtype: int64
```




```
road_condition_1
normal             2112501
construction         35337
holes                 8348
other                 7631
obstruction           7327
loose material        2954
reduced width         1649
flooded               1250
Name: at_fault, dtype: int64
```

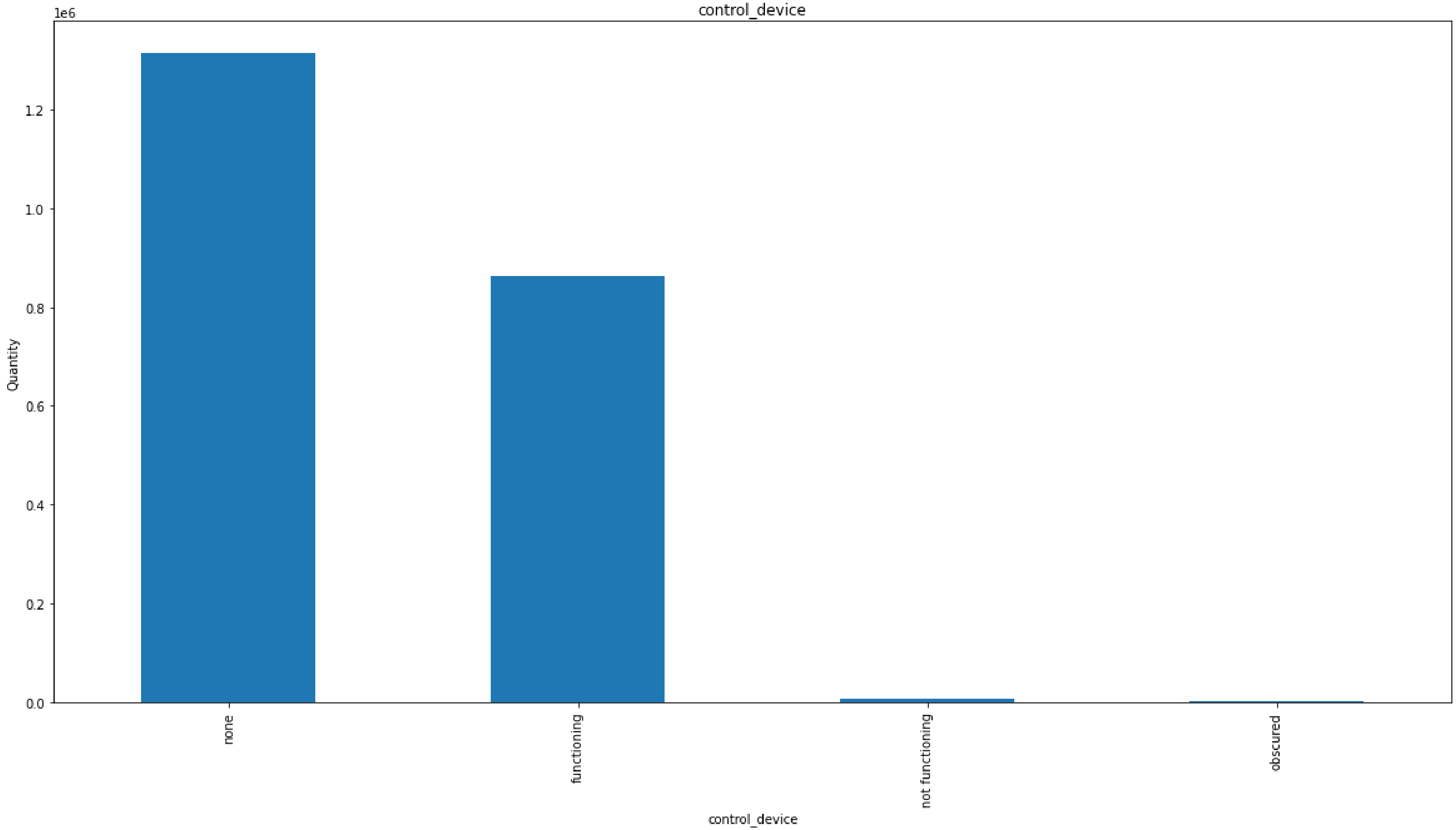



```
lighting
daylight                                      1574429
dark with street lights                        407141
dark with no street lights                     125922
dusk or dawn                                    69753
dark with street lights not functioning          6100
Name: at_fault, dtype: int64
```

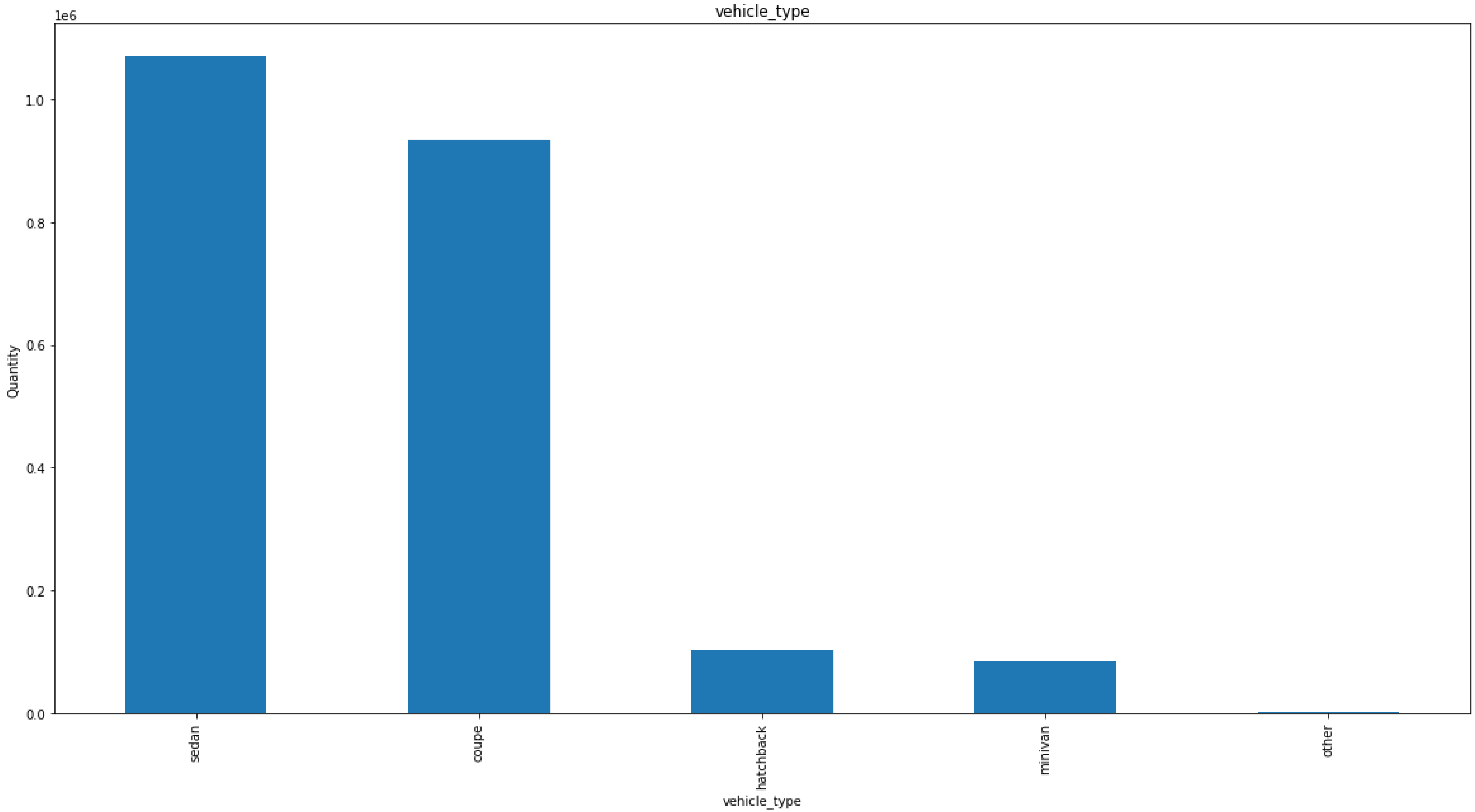


lighting
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Quantity
daylight
dark with street lights
dark with no street lights
dusk or dawn
dark with street lights not functioning
lighting

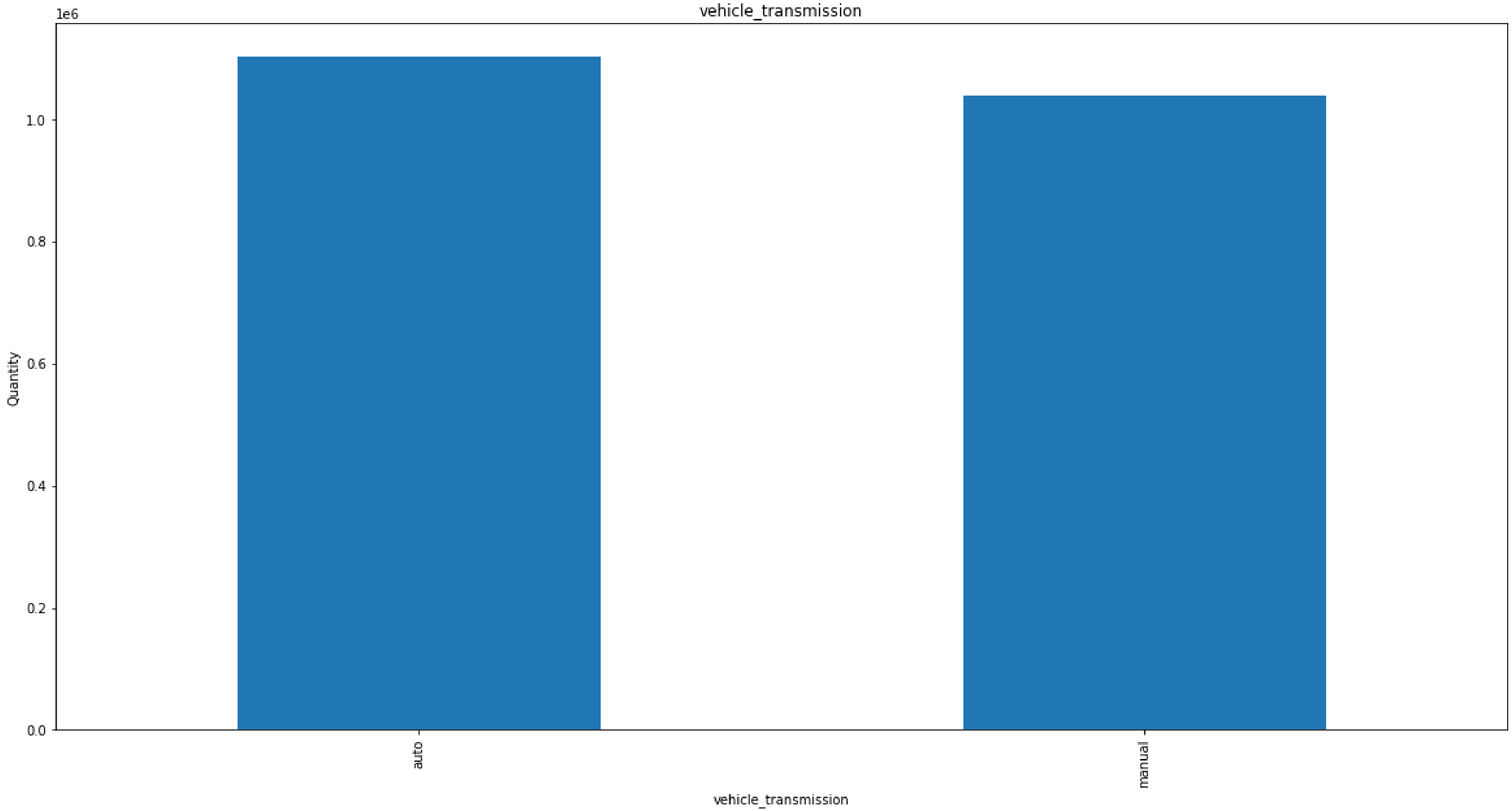
```
control_device
none               1313657
functioning         862091
not functioning       6424
obscured              1136
Name: at_fault, dtype: int64
```



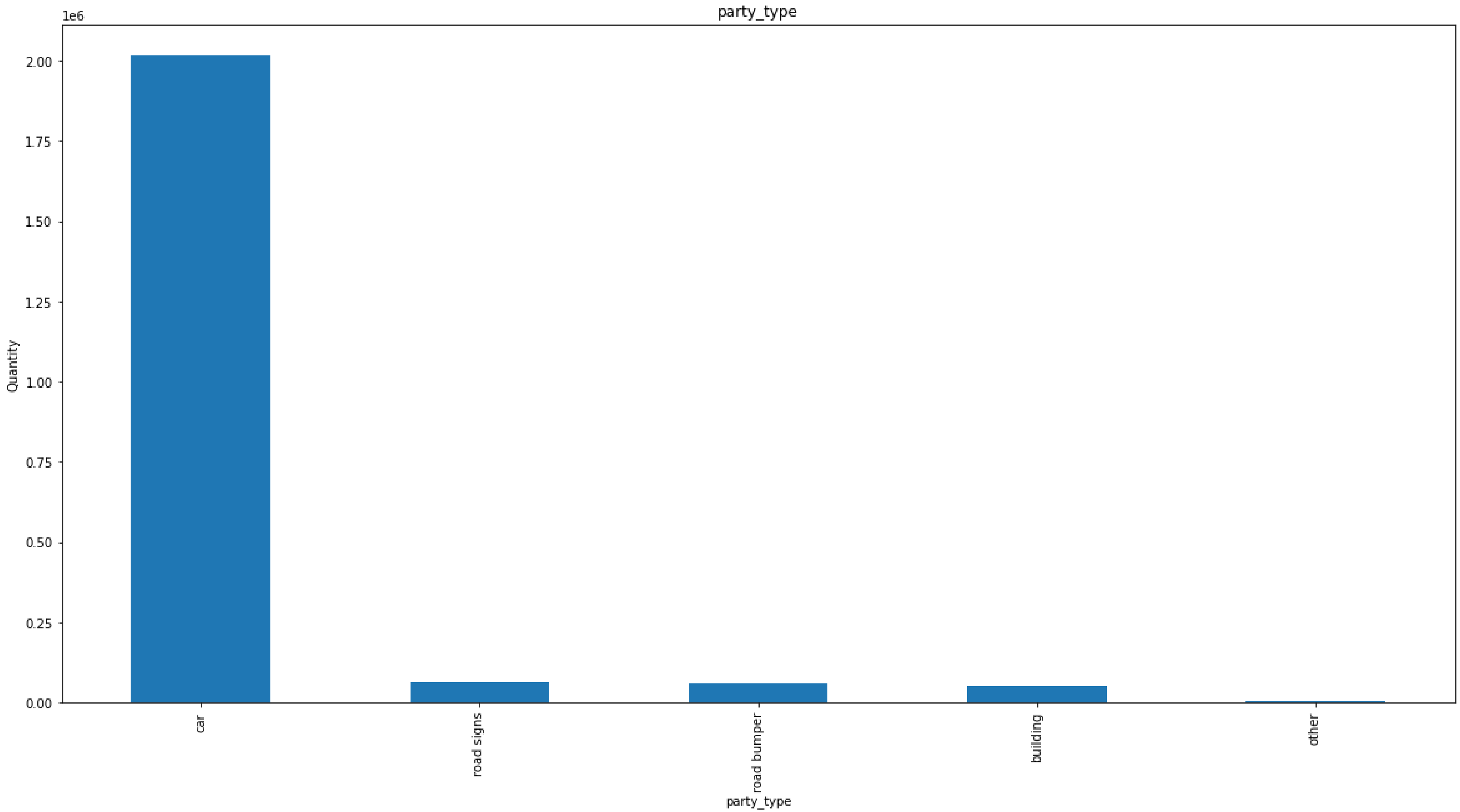
```
vehicle_type
sedan        1069935
coupe         934526
hatchback     102755
minivan        84088
other           2739
Name: at_fault, dtype: int64
```



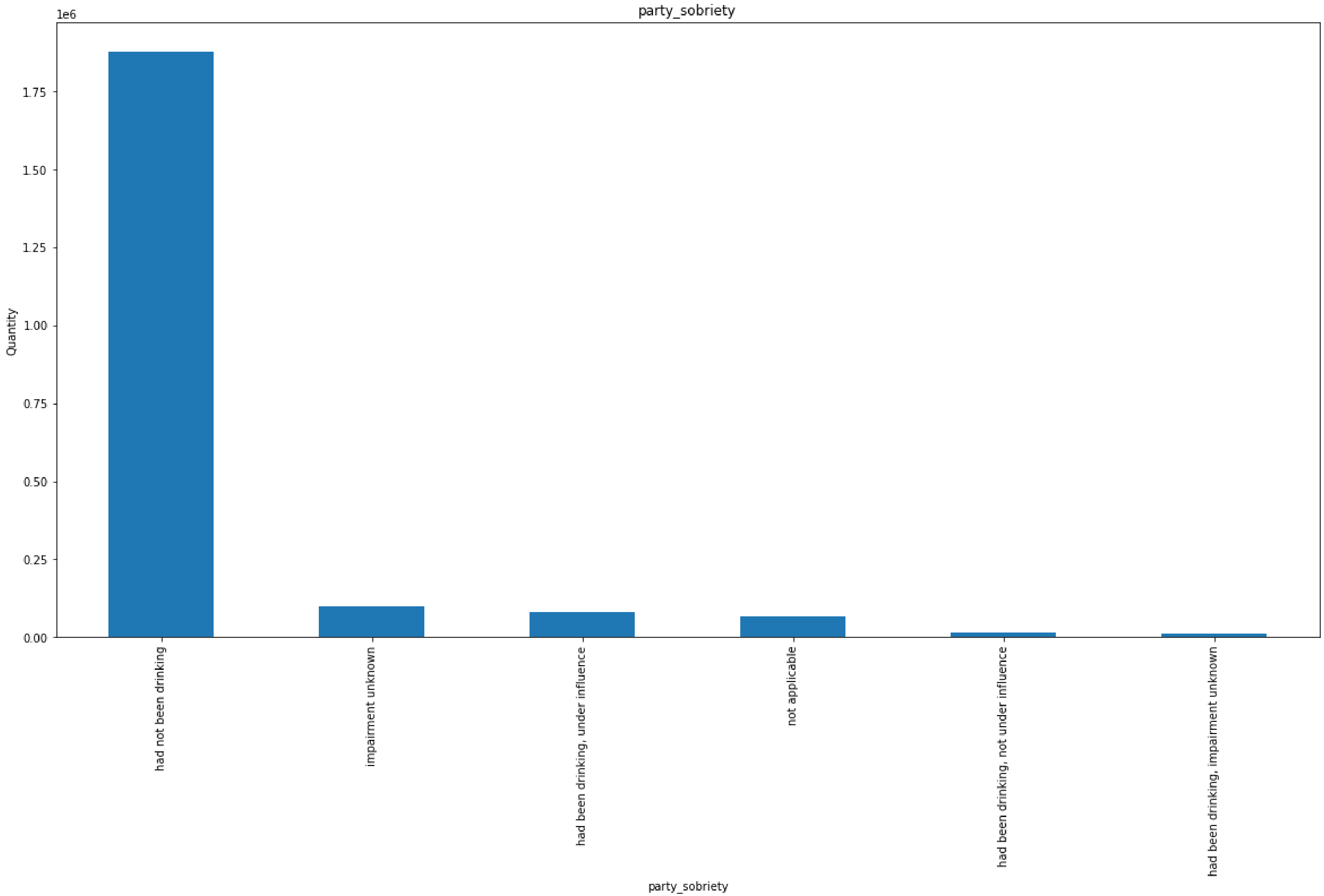

```
vehicle_transmission
auto      1103504
manual    1039858
Name: at_fault, dtype: int64
```




```
party_type
car             2014393
road signs        62564
road bumper       57367
building          52245
other              5504
Name: at_fault, dtype: int64
```

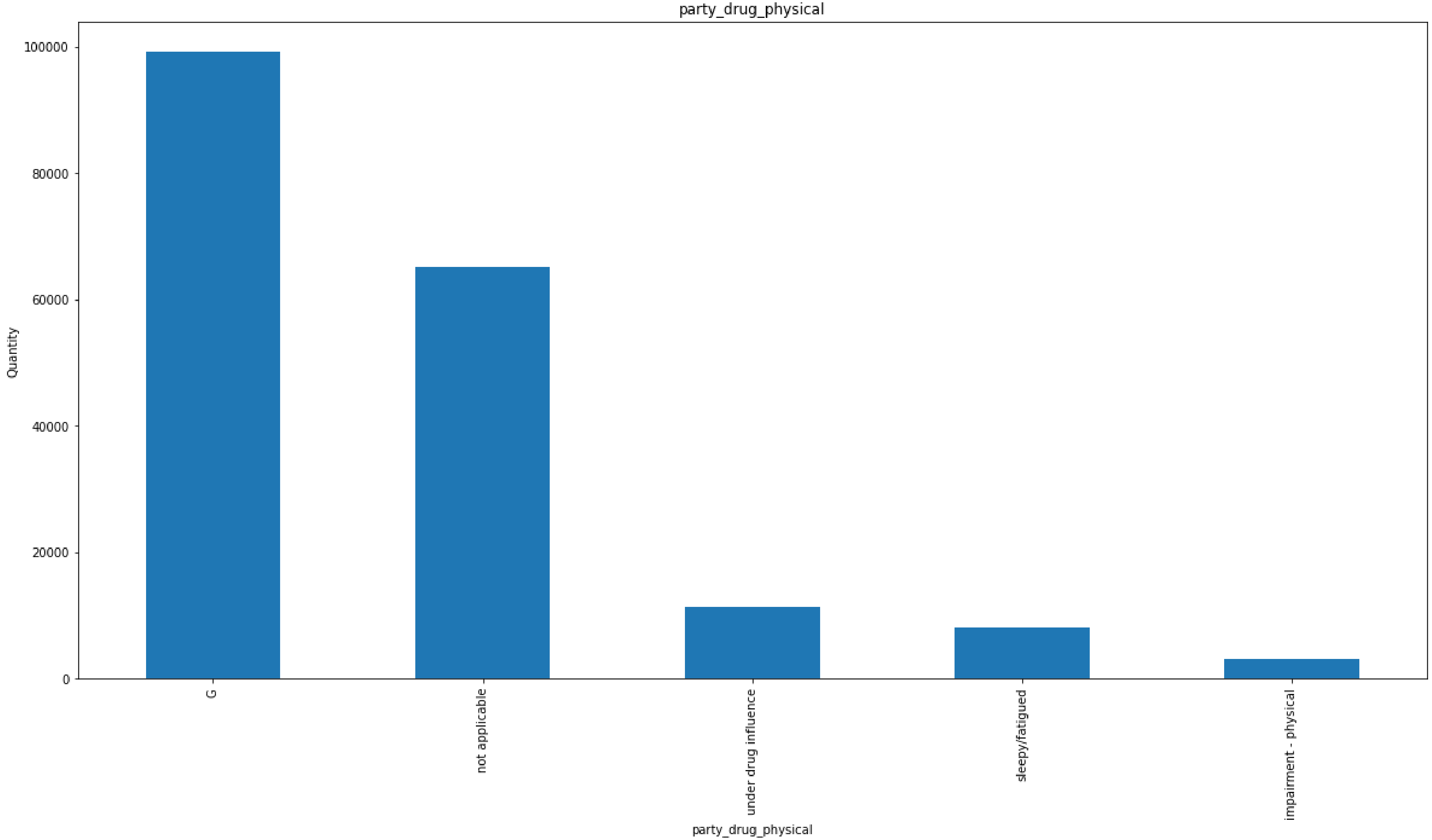



```
party_sobriety
had not been drinking                      1879053
impairment unknown                           99058
had been drinking, under influence           79412
not applicable                               65117
had been drinking, not under influence       15035
had been drinking, impairment unknown        11037
Name: at_fault, dtype: int64
```



party_sobriety
1e6
Quantity
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
had not been drinking
impairment unknown
had been drinking, under influence
not applicable
had been drinking, not under influence
had been drinking, impairment unknown
party_sobriety

```
party_drug_physical
G                         99058
not applicable            65117
under drug influence      11304
sleepy/fatigued            8030
impairment - physical      3036
Name: at_fault, dtype: int64
```



In [33]:

```
# display the statistic on numeric columns
total_df[numeric_col].describe()
```

Out[33]:

	county_city_location	distance	intersection	party_count	party_number	party_number	vehicle_age	party_number	party_number	
count	2.194043e+06	2.194043e+06	2.181354e+06	2.194043e+06	2.194043e+06	2.194043e+06	2.140681e+06	2.194043e+06	2.194043e+06	2.1940
mean	2.797314e+03	6.340588e+02	2.680239e-01	2.436112e+00	1.738596e+00	1.718074e+00	5.161942e+00	1.738596e+00	1.718074e+00	4.361
std	1.274938e+03	2.374460e+04	4.429302e-01	1.076418e+00	8.416262e-01	8.930279e-01	3.113283e+00	8.416262e-01	8.930279e-01	4.959
min	1.000000e+02	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	0.0000
25%	1.942000e+03	0.000000e+00	0.000000e+00	2.000000e+00	1.000000e+00	1.000000e+00	3.000000e+00	1.000000e+00	1.000000e+00	0.0000
50%	3.004000e+03	8.400000e+01	0.000000e+00	2.000000e+00	2.000000e+00	2.000000e+00	5.000000e+00	2.000000e+00	2.000000e+00	0.0000
75%	3.705000e+03	4.000000e+02	1.000000e+00	3.000000e+00	2.000000e+00	2.000000e+00	7.000000e+00	2.000000e+00	2.000000e+00	1.0000
max	5.802000e+03	8.363520e+06	1.000000e+00	2.700000e+01	2.700000e+01	2.700000e+01	1.610000e+02	2.700000e+01	2.700000e+01	1.0000

In [34]:

```
total_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2194043 entries, 0 to 2194042
Data columns (total 30 columns):
 #   Column                       Dtype  
---  ------                       -----  
 0   county_city_location         float64
 1   county_location              object 
 2   distance                     float64
 3   direction                    object 
 4   intersection                 float64
 5   weather_1                    object 
 6   location_type                object 
 7   collision_damage             object 
 8   party_count                  int64  
 9   primary_collision_factor     object 
 10  pcf_violation_category       object 
 11  type_of_collision            object 
 12  motor_vehicle_involved_with  object 
 13  road_surface                 object 
 14  road_condition_1             object 
 15  lighting                     object 
 16  control_device               object 
 17  collision_date               object 
 18  collision_time               object 
 19  party_number                 int64  
 20  vehicle_type                 object 
 21  vehicle_transmission         object 
 22  vehicle_age                  float64
 23  party_number                 int64  
 24  party_type                   object 
 25  at_fault                     int64  
 26  insurance_premium            float64
 27  party_sobriety               object 
 28  party_drug_physical          object 
 29  cellphone_in_use             float64
dtypes: float64(6), int64(4), object(20)
memory usage: 502.2+ MB
```

In [35]:

```
# clear the memory
del total_df
```

Building model of evaluation of driving risk

Selection of features for the model

0) case_id - to delete, has no affect on possibility of accident

1) county_city_location - to delete, has no affect on possibility of accident

2) county_location - to delete, has no affect on possibility of accident

3) distance - to delete, has no affect on possibility of accident

4) direction - to delete, has no affect on possibility of accident

5) intersection - to delete, has no affect on possibility of accident

6) weather_1 - to inclde, has an affect on possibility of accident

7) location_type - to delete, has no affect on possibility of accident

8) collision_damage - to inclde, has an affect on possibility of accident

9) party_count - to delete, has no affect on possibility of accident

10) primary_collision_factor - to delete, has no affect on possibility of accident

11) pcf_violation_category - to inclde, has an affect on possibility of accident

12) type_of_collision - to inclde, has an affect on possibility of accident

13) motor_vehicle_involved_with - to delete, has no affect on possibility of accident

14) road_surface - to inclde, has an affect on possibility of accident

15) road_condition_1 - to delete, has no affect on possibility of accident

16) lighting - to inclde, has an affect on possibility of accident

17) control_device - to inclde, has an affect on possibility of accident

18) collision_date - to delete, has no affect on possibility of accident

19) collision_time - to delete, has no affect on possibility of accident

20) id - to delete, has no affect on possibility of accident

21) case_id - to delete, has no affect on possibility of accident

22) party_number - to delete, has no affect on possibility of accident

23) vehicle_type - to delete, has no affect on possibility of accident

24) vehicle_transmission - to delete, has no affect on possibility of accident

25) vehicle_age - to inclde, has an affect on possibility of accident

26) id - to delete, has no affect on possibility of accident

27) case_id - to delete, has no affect on possibility of accident

28) party_number - to delete, has no affect on possibility of accident

29) party_type - to delete, has no affect on possibility of accident

30) at_fault - target

31) insurance_premium - to inclde, has an affect on possibility of accident

32) party_sobriety - to inclde, has an affect on possibility of accident

33) party_drug_physical - to inclde, has an affect on possibility of accident

34) cellphone_in_use - to inclde, has an affect on possibility of accident

In [36]:

```
# query to get the dataset with important features and target
query = '''
SELECT collision_damage,
```

```
       pcf_violation_category,
       type_of_collision,
       road_surface,
       lighting,
       control_device,
       vehicle_age,
       at_fault,
       insurance_premium,
       party_sobriety,
       party_drug_physical,
       cellphone_in_use
  FROM collisions AS c
       INNER JOIN Vehicles AS v
          ON c.case_id = v.case_id
          INNER JOIN Parties AS p
             ON c.case_id = p.case_id
 WHERE c.collision_damage NOT LIKE ('%%scratch%%'')
   AND p.party_type LIKE (''%%car%%')
   AND EXTRACT(YEAR FROM collision_date) = (2012.0)
'''

ttl_model_df = pd.read_sql_query(query, con=engine)
ttl_model_df.head()
```

Out[36]:

	collision_damage	pcf_violation_category	type_of_collision	road_surface	lighting	control_device	vehicle_age	at_fault	insurance_premium	party_sobr
0	fatal	pedestrian violation	other	dry	dark with street lights	none	9.0	1	60.0	had b drinking, ur influe
1	fatal	pedestrian right of way	other	dry	dark with street lights	none	8.0	0	50.0	had b drinking, ur influe
2	fatal	improper turning	rear end	dry	dark with street lights	none	3.0	0	NaN	not applic
3	fatal	improper turning	rear end	dry	dark with street lights	none	5.0	0	NaN	not applic
4	fatal	pedestrian violation	other	dry	dark with street lights	none	4.0	1	30.0	had b drinking, ur influe

In [37]:

```
ttl_model_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 246125 entries, 0 to 246124
Data columns (total 12 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   collision_damage       246125 non-null  object 
 1   pcf_violation_category 242012 non-null  object 
 2   type_of_collision      244631 non-null  object 
 3   road_surface           244160 non-null  object 
 4   lighting               245264 non-null  object 
 5   control_device         244923 non-null  object 
 6   vehicle_age            240096 non-null  float64
 7   at_fault               246125 non-null  int64  
 8   insurance_premium      229970 non-null  float64
 9   party_sobriety         241127 non-null  object 
 10  party_drug_physical    21205 non-null   object 
 11  cellphone_in_use       213747 non-null  float64
dtypes: float64(3), int64(1), object(8)
memory usage: 22.5+ MB
```

In [38]:

```
# display the unique value of column party_drug_physical
ttl_model_df['party_drug_physical'].unique()
```

Out[38]:

```
array([None, 'under drug influence', 'not applicable', 'G',
       'sleepy/fatigued', 'impairment - physical'], dtype=object)
```

In [39]:

```
# count the values by columns party_drug_physical in case of accident
ttl_model_df.groupby('party_drug_physical')['at_fault'].count().sort_values(ascending = False)
```

Out[39]:

```
party_drug_physical
G                        10898
not applicable            7102
under drug influence      1945
sleepy/fatigued            919
impairment - physical      341
Name: at_fault, dtype: int64
```

In [40]:

```
# filling the nulls
ttl_model_df['party_drug_physical'] = ttl_model_df['party_drug_physical'].fillna('G')
```

In [41]:

```
ttl_model_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 246125 entries, 0 to 246124
Data columns (total 12 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   collision_damage       246125 non-null  object 
 1   pcf_violation_category 242012 non-null  object 
 2   type_of_collision      244631 non-null  object 
 3   road_surface           244160 non-null  object 
 4   lighting               245264 non-null  object 
 5   control_device         244923 non-null  object 
 6   vehicle_age            240096 non-null  float64
 7   at_fault               246125 non-null  int64  
 8   insurance_premium      229970 non-null  float64
 9   party_sobriety         241127 non-null  object 
 10  party_drug_physical    246125 non-null  object 
 11  cellphone_in_use       213747 non-null  float64
dtypes: float64(3), int64(1), object(8)
memory usage: 22.5+ MB
```

Deletion of nulls

In [42]:
```
ttl_model_df = ttl_model_df.dropna()
```

In [43]:
```
target = ttl_model_df['at_fault']
```

In [44]:
```
# Selection of categorical and numeric columns
numeric_col = list(ttl_model_df.select_dtypes(include=['int64', 'float64']).columns[:])
numeric_col.remove('at_fault')
categorical_col = list(ttl_model_df.select_dtypes(include=['object']).columns[:])
print('Numerical columns:', numeric_col,'\n')
print('Categorical columns:','\n',categorical_col,'\n')
```

```
Numerical columns: ['vehicle_age', 'insurance_premium', 'cellphone_in_use']

Categorical columns:
 ['collision_damage', 'pcf_violation_category', 'type_of_collision', 'road_surface', 'lighting', 'control_device', 'party_sobrie
ty', 'party_drug_physical']
```

Splitting the data on samples

In [45]:
```
X_train_for_coding, X_valid_for_coding, y_train, y_valid = train_test_split(ttl_model_df.drop(columns = ['at_fault']), ttl_model_
```

Rescaling the numeric columns

In [46]:
```
numeric_col
```

Out[46]:
```
['vehicle_age', 'insurance_premium', 'cellphone_in_use']
```

In [47]:
```
scaler = MinMaxScaler()
```

In [48]:
```
scaler.fit(X_train_for_coding[numeric_col])
X_train_numeric = pd.DataFrame(scaler.transform(X_train_for_coding[numeric_col]),columns = X_train_for_coding[numeric_col].colum
X_valid_numeric = pd.DataFrame(scaler.transform(X_valid_for_coding[numeric_col]),columns = X_valid_for_coding[numeric_col].colum
```

In [49]:
```
X_train_numeric.head(10)
```

Out[49]:

	vehicle_age	insurance_premium	cellphone_in_use
0	0.012422	0.504762	0.0
1	0.043478	0.714286	0.0
2	0.074534	0.371429	0.0
3	0.031056	0.266667	0.0
4	0.049689	0.314286	0.0
5	0.006211	0.342857	0.0
6	0.037267	0.390476	0.0
7	0.049689	0.380952	0.0
8	0.018634	0.361905	0.0
9	0.043478	0.438095	0.0

In [50]:
```
X_train_numeric.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 116333 entries, 0 to 116332
Data columns (total 3 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   vehicle_age        116333 non-null  float64
 1   insurance_premium  116333 non-null  float64
 2   cellphone_in_use   116333 non-null  float64
dtypes: float64(3)
memory usage: 2.7 MB
```

In [51]:
```
X_valid_numeric.head(10)
```

Out[51]:

	vehicle_age	insurance_premium	cellphone_in_use
0	0.068323	0.342857	0.0
1	0.012422	0.171429	0.0
2	0.043478	0.447619	0.0
3	0.031056	0.380952	0.0
4	0.024845	0.304762	0.0
5	0.024845	0.504762	0.0
6	0.031056	0.361905	0.0
7	0.031056	0.409524	0.0
8	0.012422	0.619048	0.0
9	0.031056	0.657143	0.0

In [52]:
```
X_valid_numeric.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77556 entries, 0 to 77555
Data columns (total 3 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   vehicle_age        77556 non-null  float64
 1   insurance_premium  77556 non-null  float64
 2   cellphone_in_use   77556 non-null  float64
dtypes: float64(3)
memory usage: 1.8 MB
```

Encoding of categorical features

In [53]:
```
ohe = OneHotEncoder()
X_train_categorical = ohe.fit_transform(X_train_for_coding[categorical_col]).toarray()
X_valid_categorical = ohe.transform(X_valid_for_coding[categorical_col]).toarray()
```

In [54]:
```
X_train_categorical = pd.DataFrame(X_train_categorical)
X_valid_categorical = pd.DataFrame(X_valid_categorical)
```

In [55]:
```
X_train_categorical.head()
```

Out[55]:

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	53	54	55	56	57
0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

5 rows × 58 columns

In [56]:
```
X_valid_categorical.head()
```

Out[56]:

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	53	54	55	56	57
0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

5 rows × 58 columns

In [57]:

```
# merge the categorical and numeric prepared features
X_train = X_train_numeric.join(X_train_categorical)
X_valid = X_valid_numeric.join(X_valid_categorical)
```

In [58]:

```
X_train.head()
```

Out[58]:

	vehicle_age	insurance_premium	cellphone_in_use	0	1	2	3	4	5	6	...	48	49	50	51	52	53	54	55	56	57
0	0.012422	0.504762	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.043478	0.714286	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.074534	0.371429	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.031056	0.266667	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.049689	0.314286	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

5 rows × 61 columns

In [59]:

```
X_valid.head()
```

Out[59]:

	vehicle_age	insurance_premium	cellphone_in_use	0	1	2	3	4	5	6	...	48	49	50	51	52	53	54	55	56	57
0	0.068323	0.342857	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.012422	0.171429	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.043478	0.447619	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.031056	0.380952	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.024845	0.304762	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

5 rows × 61 columns

In [60]:

```
# split the valid data on valid and test samples
X_test, X_valid, y_test, y_valid = train_test_split(X_valid, y_valid, test_size=0.5, random_state=42)
```

Building of simple neural network

The roc auc score to be used for models evaluation. Auc roc selected due to the fact the it's not only allows to evaulate the result, moreover it allows to evaluate the probabily of prediction.

In [61]:

```
# transformation of data to tensors

X_train_tensor = torch.FloatTensor(X_train.values)
X_test_tensor = torch.FloatTensor(X_test.values)
y_train_tensor = torch.FloatTensor(y_train.values)
y_test_tensor = torch.FloatTensor(y_test.values)
X_valid_tensor = torch.FloatTensor(X_valid.values)
y_valid_tensor = torch.FloatTensor(y_valid.values)
```

In [62]:

```
# building of neural network

torch.manual_seed(1234)
input_size = 61
hidden_size_1 = 15
hidden_size_2 = 8
output_size = 1

class NeuralNet(nn.Module):
```

```python
    def __init__(self, input_size, hidden_size_1, hidden_size_2, output_size):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size_1)
        self.act1 = nn.Tanh()
        self.fc2 = nn.Linear(hidden_size_1, hidden_size_2)
        self.act2 = nn.Tanh()
        self.fc3 = nn.Linear(hidden_size_2, output_size)
        self.act3 = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.act1(x)
        x = self.fc2(x)
        x = self.act2(x)
        x = self.fc3(x)
        x = self.act3(x)
        return x

nn_model = NeuralNet(input_size, hidden_size_1, hidden_size_2, output_size)
```

In [63]:

```python
# training of nn_model and predict the target

optimizer = torch.optim.Adam(nn_model.parameters(),lr=0.0015)

loss = torch.nn.BCELoss()

num_epochs = 1000

for epoch in range(num_epochs):
    optimizer.zero_grad()
    preds = nn_model.forward(X_train_tensor).flatten()
    loss_value = loss(preds,y_train_tensor)
    loss_value.backward()
    optimizer.step()
    if (epoch % 100 == 0) or (epoch == 1000) :
            print(loss_value)
            nn_model.eval(),
            nn_model_preds = nn_model.forward(X_valid_tensor).flatten()
            accuracy = (torch.round(nn_model_preds) == y_valid_tensor).float().mean().data
            print(accuracy)
```

```
tensor(0.7081, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.4425)
tensor(0.6330, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.6331)
tensor(0.6076, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.6414)
tensor(0.6031, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.6425)
tensor(0.5998, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.6435)
tensor(0.5971, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.6463)
tensor(0.5946, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.6480)
tensor(0.5922, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.6514)
tensor(0.5900, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.6538)
tensor(0.5878, grad_fn=<BinaryCrossEntropyBackward0>)
tensor(0.6549)
```

In [64]:

```
nn_preds = torch.round(nn_model_preds, decimals=0).detach().numpy()
```

In [65]:

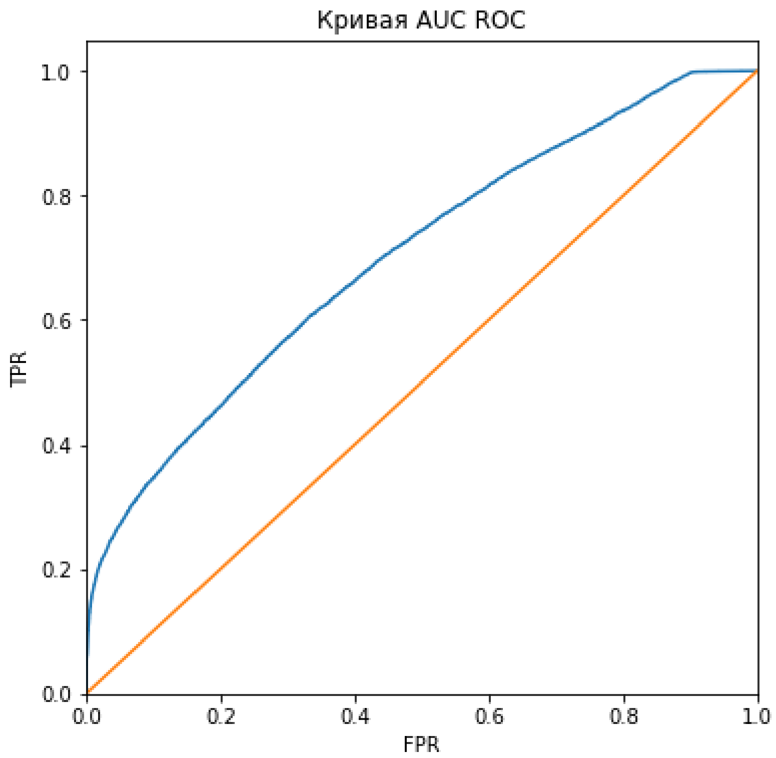
```
# display the auc roc score
roc_auc_score_nn = roc_auc_score(y_valid, nn_preds, average=None)
roc_auc_score_nn
```

Out[65]:

```
0.6242019244517162
```

In [66]:

```
# plot the auc roc curve

precision, recall, thresholds = roc_curve(y_valid, nn_model_preds.detach().numpy())

plt.figure(figsize=(6, 6))
plt.step( precision,recall, where='post')
plt.plot([0.0,1.0],[0.0,1.0])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Кривая AUC ROC')
plt.show()
```



Training of random forest model

In [67]:
```
parameters = { 'n_estimators': range (10, 41, 10),
               'max_depth': range (1,40,10)}

model_rf  = RandomizedSearchCV(RandomForestClassifier(random_state=12345),parameters,scoring='roc_auc',random_state=12345)
```

In [68]:
```
model_rf.fit(X_train, y_train)
```

Out[68]:
```
RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [69]:
```
print(model_rf.best_params_)
roc_auc_cv_rf = model_rf.best_score_
model_rf.best_score_
```

```
{'n_estimators': 40, 'max_depth': 21}
```

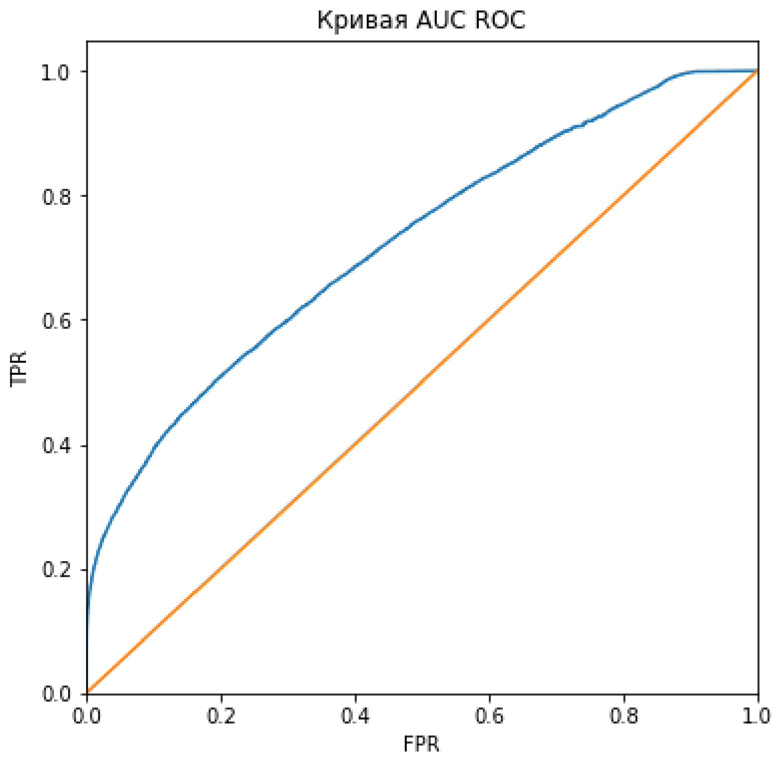
Out[69]:
```
0.7239854982152952
```

In [70]:
```
predictions_rf = model_rf.predict(X_valid)
```

In [71]:
```
# display auc roc score
probabilities_rf = model_rf.predict_proba(X_valid)
roc_auc_rf = roc_auc_score(y_valid,probabilities_rf[:, 1])
roc_auc_rf
```

Out[71]:
```
0.7224588632217692
```

In [72]:
```
# plot the auc roc curve
precision, recall, thresholds = roc_curve(y_valid, probabilities_rf[:, 1])


plt.figure(figsize=(6, 6))
plt.step( precision,recall, where='post')
plt.plot([0.0,1.0],[0.0,1.0])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Кривая AUC ROC')
plt.show()
```



Training of decision tree model

In [73]:
```
parameters = { 'max_depth': range (1,100,10),
              'max_leaf_nodes': list(range(2, 100)),
               'min_samples_split': [2, 3, 4]}

model_dt = RandomizedSearchCV(DecisionTreeClassifier(random_state=12345), parameters, scoring ='roc_auc')
```

In [74]:
```
model_dt.fit(X_train, y_train)
```

Out[74]:
```
RandomizedSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

In [75]:
```
print(model_dt.best_params_)
roc_auc_cv_dt = model_dt.best_score_
model_dt.best_score_
```

```
{'min_samples_split': 2, 'max_leaf_nodes': 89, 'max_depth': 31}
```

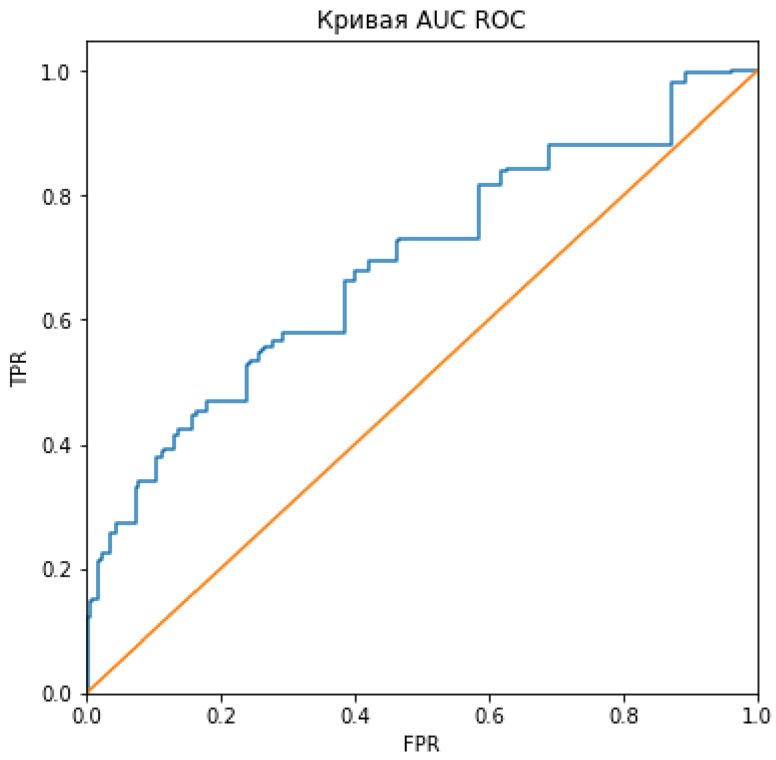
Out[75]:
```
0.7170041072103599
```

In [76]:
```
predictions_dt = model_dt.predict(X_valid)
```

In [77]:
```
# display the auc roc score
probabilities_dt = model_dt.predict_proba(X_valid)
roc_auc_dt = roc_auc_score(y_valid,probabilities_dt[:, 1])
roc_auc_dt
```

Out[77]:
```
0.7122762133170126
```

In [78]:
```
# plot the auc roc curve
precision, recall, thresholds = roc_curve(y_valid, probabilities_dt[:, 1])

plt.figure(figsize=(6, 6))
plt.step( precision,recall, where='post')
plt.plot([0.0,1.0],[0.0,1.0])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Кривая AUC ROC')
plt.show()
```



Selection of best model

In [79]:
```
table = ['model_nn','model_rf','model_dt']
```

In [80]:
```
result = [roc_auc_score_nn,roc_auc_rf,roc_auc_dt]
```

In [81]:
```
df_result = pd.DataFrame(table,columns = ['model_name'])
```

In [82]:
```
df_result['roc_auc_score'] = result
```

In [83]:
```
df_result.sort_values('roc_auc_score',ascending = False)
```

Out[83]:

	model_name	roc_auc_score
1	model_rf	0.722459
2	model_dt	0.712276
0	model_nn	0.624202

Analysis of accident factors impact

best model testing

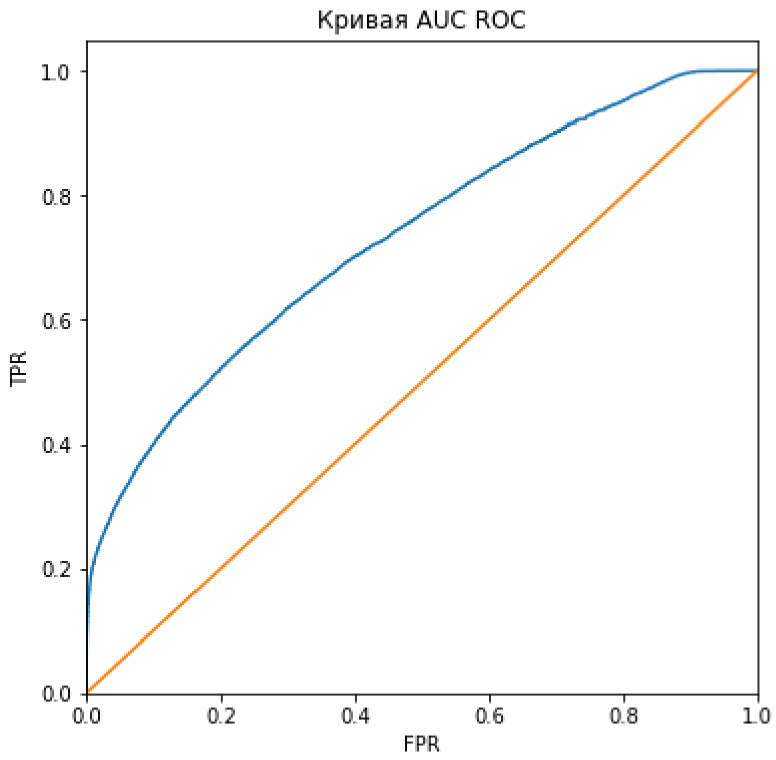
In [84]:
```
predictions_rf_test = model_rf.predict(X_test)
```

In [85]:
```
probabilities_rf_test = model_rf.predict_proba(X_test)[:, 1]
```

In [86]:
```
# display the auc roc on test sample
roc_auc_rf_test = roc_auc_score(y_test,probabilities_rf_test)
roc_auc_rf_test
```

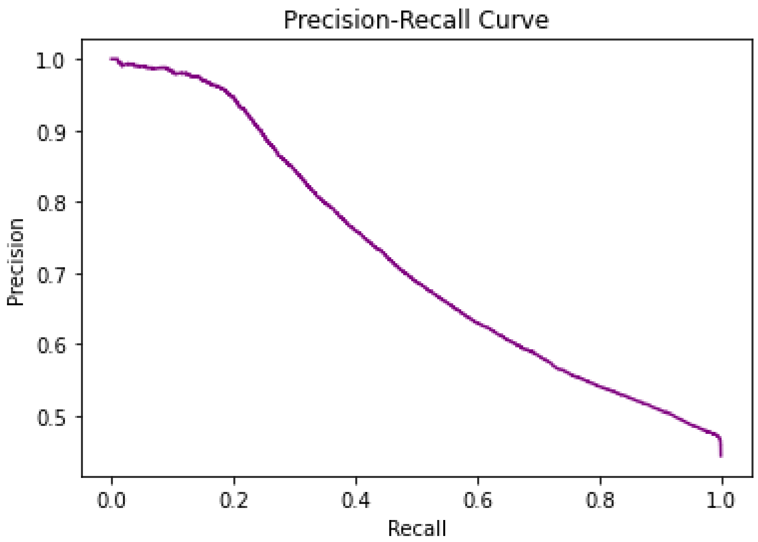
Out[86]:
```
0.7314310940350061
```

In [87]:
```
# plot the auc roc curve
precision, recall, thresholds = roc_curve(y_test, probabilities_rf_test)

plt.figure(figsize=(6, 6))
plt.step( precision,recall, where='post')
plt.plot([0.0,1.0],[0.0,1.0])
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Кривая AUC ROC')
plt.show()
```



Plot the precision-recall curve

In [88]:
```
precision, recall, thresholds = precision_recall_curve(y_test, probabilities_rf_test)

fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
plt.show()
```



Analysis of violations which has impact on the accident possibility

In [89]:
```
fault_df = ttl_model_df
```

In [90]:
```
fault_df = fault_df.drop(columns = ['collision_damage','type_of_collision'])
```

In [91]:
```
fault_df['at_fault'].sum()
```

Out[91]:
```
85547
```

In [92]:
```
# pivot table qounts the quantity of accidents per violation
factors_count_df = pd.pivot_table(fault_df,values='at_fault',index=['pcf_violation_category'], aggfunc=np.sum).reset_index()
```

In [93]:
```
factors_count_df.columns= ["fault_factor", "count"]
```

In [94]:
```
# Loop for index change
temp = []
for i in factors_count_df.index:
    temp.append('pcf_violation_category')
factors_count_df.index = temp
```

In [95]:
```
factors_count_df
```

Out[95]:

	fault_factor	count
pcf_violation_category	automobile right of way	12897
pcf_violation_category	brakes	19
pcf_violation_category	dui	6770
pcf_violation_category	fell asleep	1
pcf_violation_category	following too closely	2029
pcf_violation_category	hazardous parking	45
pcf_violation_category	impeding traffic	54
pcf_violation_category	improper passing	599
pcf_violation_category	improper turning	11444
pcf_violation_category	lights	19
pcf_violation_category	other equipment	96
pcf_violation_category	other hazardous violation	735
pcf_violation_category	other improper driving	309
pcf_violation_category	other than driver (or pedestrian)	0
pcf_violation_category	pedestrian right of way	1254
pcf_violation_category	pedestrian violation	1394
pcf_violation_category	speeding	32611
pcf_violation_category	traffic signals and signs	5613
pcf_violation_category	unknown	51
pcf_violation_category	unsafe lane change	5054
pcf_violation_category	unsafe starting or backing	2107
pcf_violation_category	wrong side of road	2446

In [96]:

```
# Loop for cration of column with foult factor anf calculation of percentage of it
for i in fault_df.columns:
```

```python
    if i == 'at_fault':
        print()
    elif i == 'pcf_violation_category':
        print()
    else:
        temp_df  = pd.pivot_table(fault_df,values='at_fault',
                                                  index=[i],
                                                  aggfunc=np.sum).reset_index()
        temp_df.columns= ["fault_factor", "count"]
        temp_list = []
        for n in temp_df.index:
            temp_list.append(i)
        temp_df.index = temp_list
        factors_count_df = factors_count_df.append(temp_df)
```

In [97]:

```python
factors_count_df['percentage'] = round(factors_count_df['count'] / fault_df['at_fault'].sum()*100,0)
```

In [98]:

```python
factors_count_df.sort_values('count',ascending = False).head(30)
```

Out[98]:

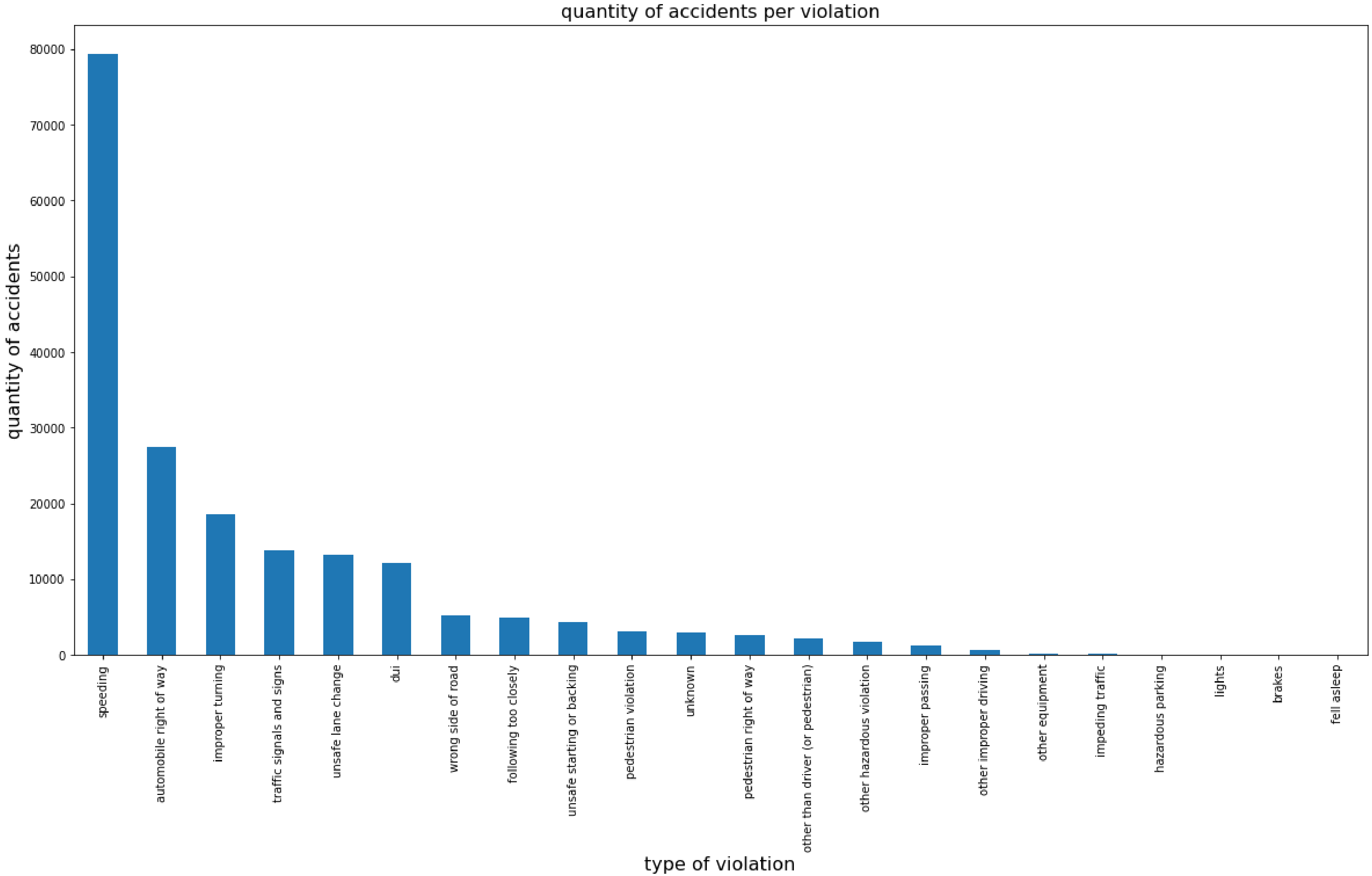
	fault_factor	count	percentage
party_drug_physical	G	84028	98.0
cellphone_in_use	0.0	83537	98.0
road_surface	dry	76689	90.0
party_sobriety	had not been drinking	75215	88.0
lighting	daylight	61252	72.0
control_device	none	54866	64.0
pcf_violation_category	speeding	32611	38.0
control_device	functioning	30430	36.0
vehicle_age	3.0	15591	18.0
lighting	dark with street lights	14903	17.0
pcf_violation_category	automobile right of way	12897	15.0
pcf_violation_category	improper turning	11444	13.0
vehicle_age	4.0	10935	13.0
vehicle_age	5.0	9118	11.0
vehicle_age	2.0	8406	10.0
road_surface	wet	8336	10.0
party_sobriety	had been drinking, under influence	7150	8.0
vehicle_age	7.0	7059	8.0
vehicle_age	6.0	7009	8.0
pcf_violation_category	dui	6770	8.0
vehicle_age	8.0	6698	8.0
lighting	dark with no street lights	6524	8.0
pcf_violation_category	traffic signals and signs	5613	7.0
vehicle_age	9.0	5216	6.0

	fault_factor	count	percentage
pcf_violation_category	unsafe lane change	5054	6.0
insurance_premium	19.0	3759	4.0
insurance_premium	20.0	3569	4.0
vehicle_age	10.0	3535	4.0
vehicle_age	0.0	3506	4.0
insurance_premium	21.0	3397	4.0

The highest quantity of accident happends due to the violation of speed limit - 38% случаев.

In [99]:
```
# plottng of quantity of accident per violation
fault_df.groupby('pcf_violation_category')['at_fault'].count().sort_values(ascending = False).plot(kind = 'bar',figsize = (20,10
plt.title('quantity of accidents per violation', fontsize='16')
plt.xlabel('type of violation', fontsize='16')
plt.ylabel('quantity of accidents', fontsize='16')
```

Out[99]:
```
Text(0, 0.5, 'quantity of accidents')
```



Proposal: for the reduction of quantity of accident it's possible to develop a software which will not allow user to exceed the speed limit.

General Conclusion

The best models for the possibility accident pridiction is random forest, with auc roc score - 0.72

Deevelopment of accident prediction system is possible, but the quantity of factor with affect of the accident is very high. The easiest way to set the parameters which will track the drivers profile and restrict the possibily of car rental in case of any specific violations.

Addition factor which also shall be considered - drivers age, experience, quantity of accidents, etc. (some of data s possible to get from insurance companies in case of pertnership.)