Avito demand prediction

Avito, Russia's largest classified advertisements website, is deeply familiar with this problem. Sellers on their platform sometimes feel frustrated with both too little demand (indicating something is wrong with the product or the product listing) or too much demand (indicating a hot item with a good description was underpriced).

Business objective

To predict the deal probability of a product given an online advertisement, based on it's full description (title, description, images, etc.), its context (geographically where it was posted, similar ads already posted) and historical demand for similar ads in similar contexts.

Data Information

1. train.csv

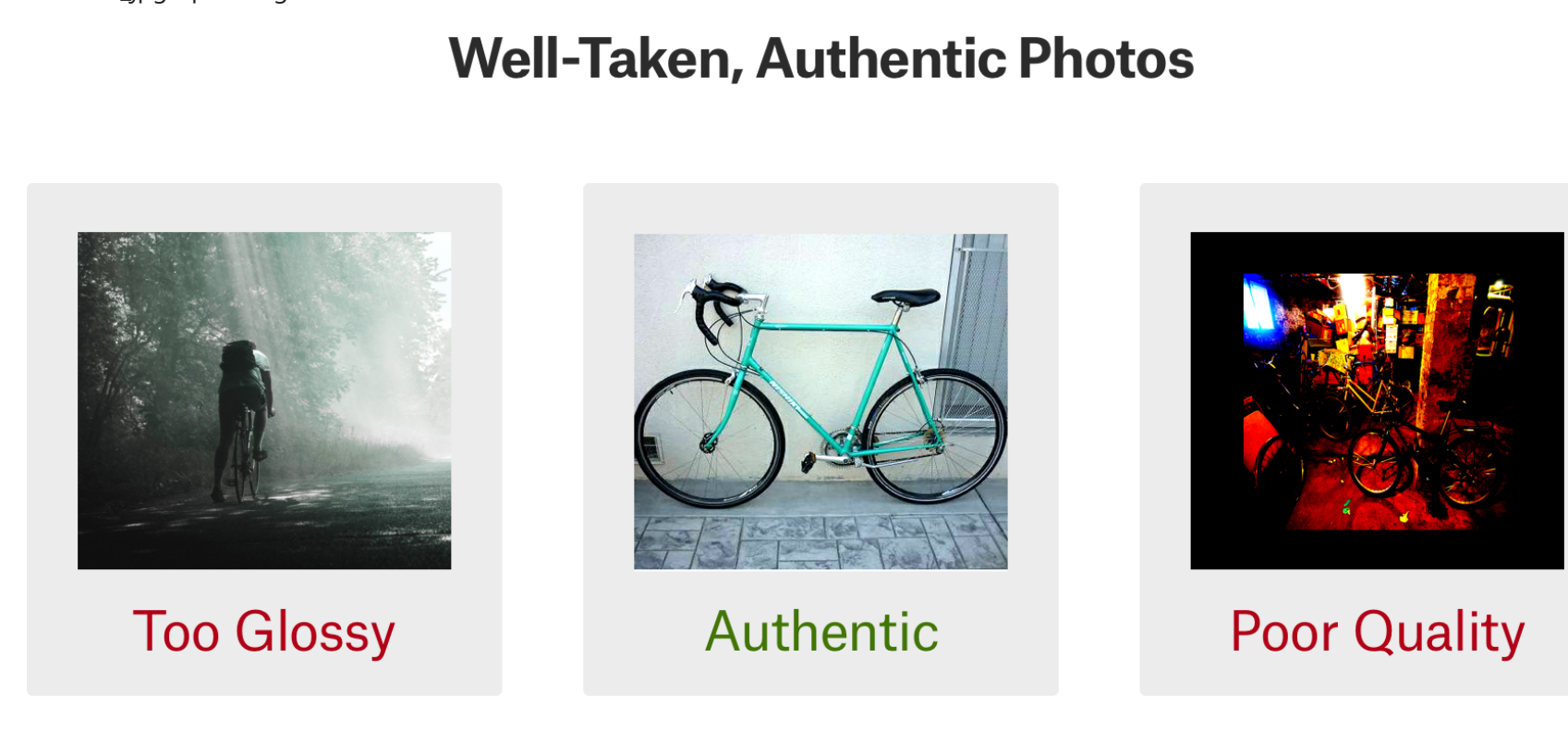
- item_id - Ad id.
- user_id - User id.
- region - Ad region.
- city - Ad city.
- parent_category_name - Top level ad category as classified by Avito's ad model.
- category_name - Fine grain ad category as classified by Avito's ad model.
- param_1 - Optional parameter from Avito's ad model.
- param_2 - Optional parameter from Avito's ad model.
- param_3 - Optional parameter from Avito's ad model.
- title - Ad title.
- description - Ad description.
- price - Ad price.
- item_seq_number - Ad sequential number for user.
- activation_date- Date ad was placed.
- user_type - User type.
- image - Id code of image. Ties to a jpg file in train_jpg. Not every ad has an image.
- image_top_1 - Avito's classification code for the image.
- deal_probability - The target variable. This is the likelihood that an ad actually sold something. It's not possible to verify every transaction with certainty, so this column's value can be any float from zero to one.

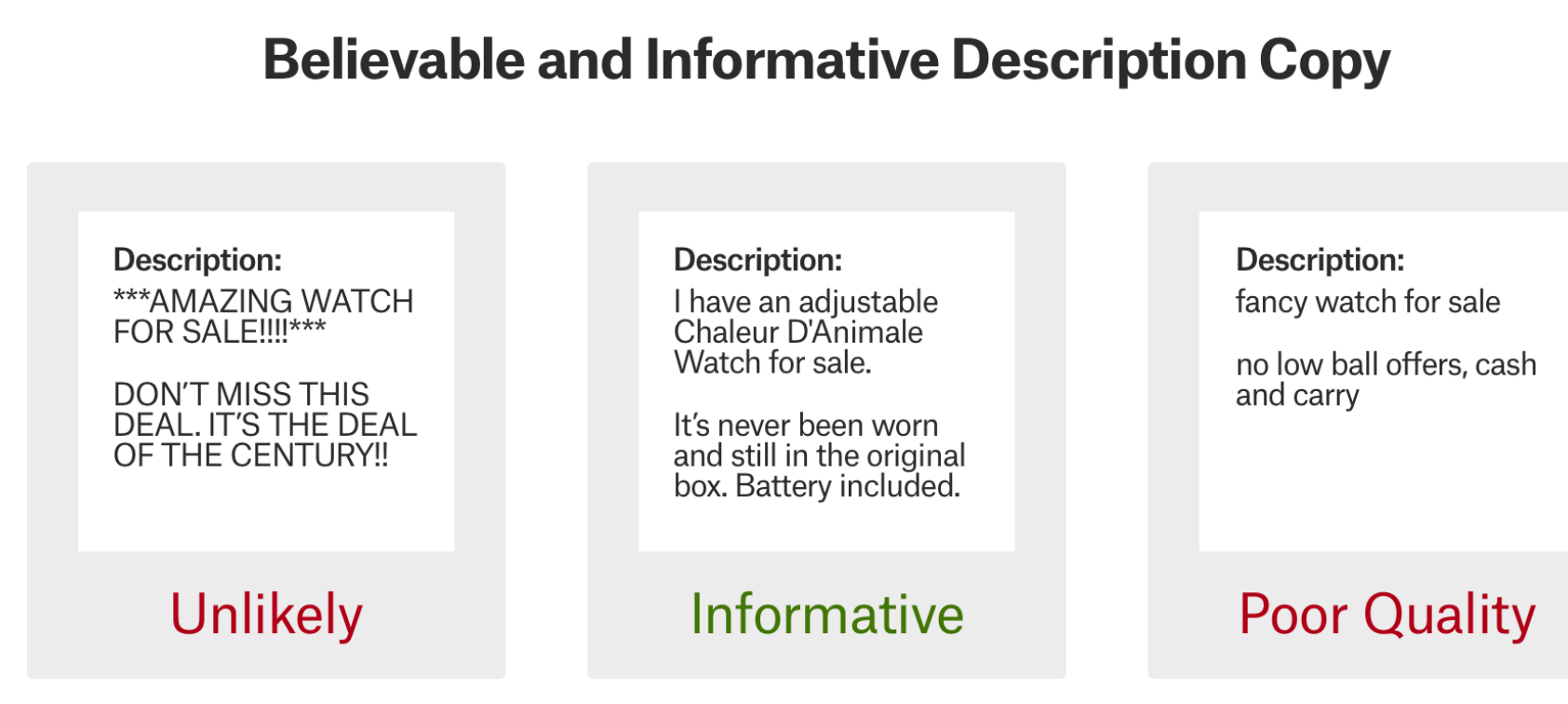
2. test.csv

- Test data. Same schema as the train data, minus deal_probability.

3. Ad images

- train_jpg.zip - Images from the ads in train.csv.
- test_jpg.zip - Images from the ads in test.csv.





3. Other files

- train_active.csv - Supplemental data from ads that were displayed during the same period as train.csv. Same schema as * the train data minus deal_probability, image, and image_top_1.
- test_active.csv - Supplemental data from ads that were displayed during the same period as test.csv. Same schema as the * train data minus deal_probability, image, and image_top_1.
- periods_train.csv - Supplemental data showing the dates when the ads from train_active.csv were activated and when they where displayed.
  - item_id - Ad id. Maps to an id in train_active.csv. IDs may show up multiple times in this file if the ad was renewed.
  - activation_date - Date the ad was placed.
  - date_from - First day the ad was displayed.
  - date_to - Last day the ad was displayed.
- periods_test.csv - Supplemental data showing the dates when the ads from test_active.csv were activated and when they where displayed. Same schema as periods_train.csv, except that the item ids map to an ad in test_active.csv.
- sample_submission.csv - A sample submission in the correct format.

Evaluation metric

- RMSE (Root mean squared error)

$$RMSE=\sqrt{\frac{\sum_{i=1}^{N}(Predicted_i-Actual_i)^2}{N}}$$

- Where,
  - predicted = The predicted value for the ith observation.
  - actual = The observed(actual) value for the ith observation
  - N = Total number of observations.

In [1]:
```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
train_df=pd.read_pickle('data/train_processed.pkl').drop(['param_1_eng','param_2_eng','param_3_eng'],axis=1)
```

In [3]:
```
train_df
```

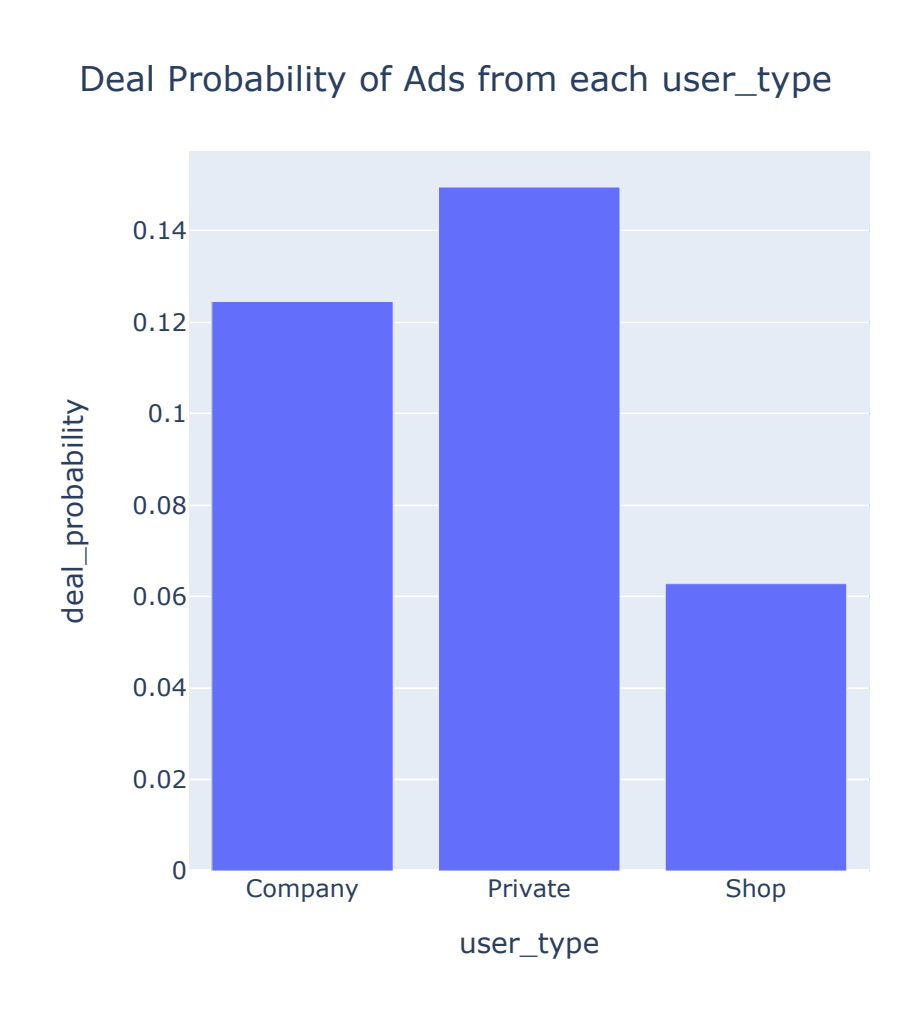
Out[3]:

	item_id	user_id	price	item_seq_number	activation_date	user_type	imag
0	b912c3c6a6ad	e00f8ff2eaf9	400.0	2	2017-03-28	Private	d10c7e016e03247a3bf2d13348fe959fe6f436c1caf64c.
1	2dac0150717d	39aeb48f0017	3000.0	19	2017-03-26	Private	79c9392cc51a9c81c6eb91eceb8e552171db39d7142700.
2	ba83aefab5dc	91e2f88dd6e3	4000.0	9	2017-03-20	Private	b7f250ee3f39e1fedd77c141f273703f4a9be59db4b48a.
3	02996f1dd2ea	bf5cccea572d	2200.0	286	2017-03-25	Company	e6ef97e0725637ea84e3d203e82dadb43ed3cc0a1c8413.
4	7c90be56d2ab	ef50846afc0b	40000.0	3	2017-03-16	Private	54a687a3a0fc1d68aed99bdaaf551c5c70b761b16fd0a2.
...	...	...	...	...	...	...	.
1503419	5e6b7f0f3f65	dd05217165ea	300.0	7	2017-03-20	Private	2da79294c6f23f61d4bf6a8f4034f87e2f318b7e32c68b.
1503420	d1f0910d2126	dc23ef451c8a	200.0	1	2017-03-28	Private	33e13a67b441486ebe48e1459382caa620c4d1802aad1c.
1503421	bc04866bc803	66a7016a5995	NaN	50	2017-03-21	Company	2ae822ca21cff9465e430c68a1df31c5cda658cbc00eb1.
1503422	f782f2ad9349	2d3ac1f5a7b3	350.0	33	2017-03-22	Private	6b4285c22388a5dae37c3d390d3e1b174f7a980d9e3919.
1503423	9ad3b7bff1db	f3342ccd95c8	1500.0	18	2017-03-21	Private	87cca5889b96b0368bffbe7cdb3f0cc5fa660f33be9fd0.

1503424 rows × 15 columns

Deal Probability of Ads from each user_type

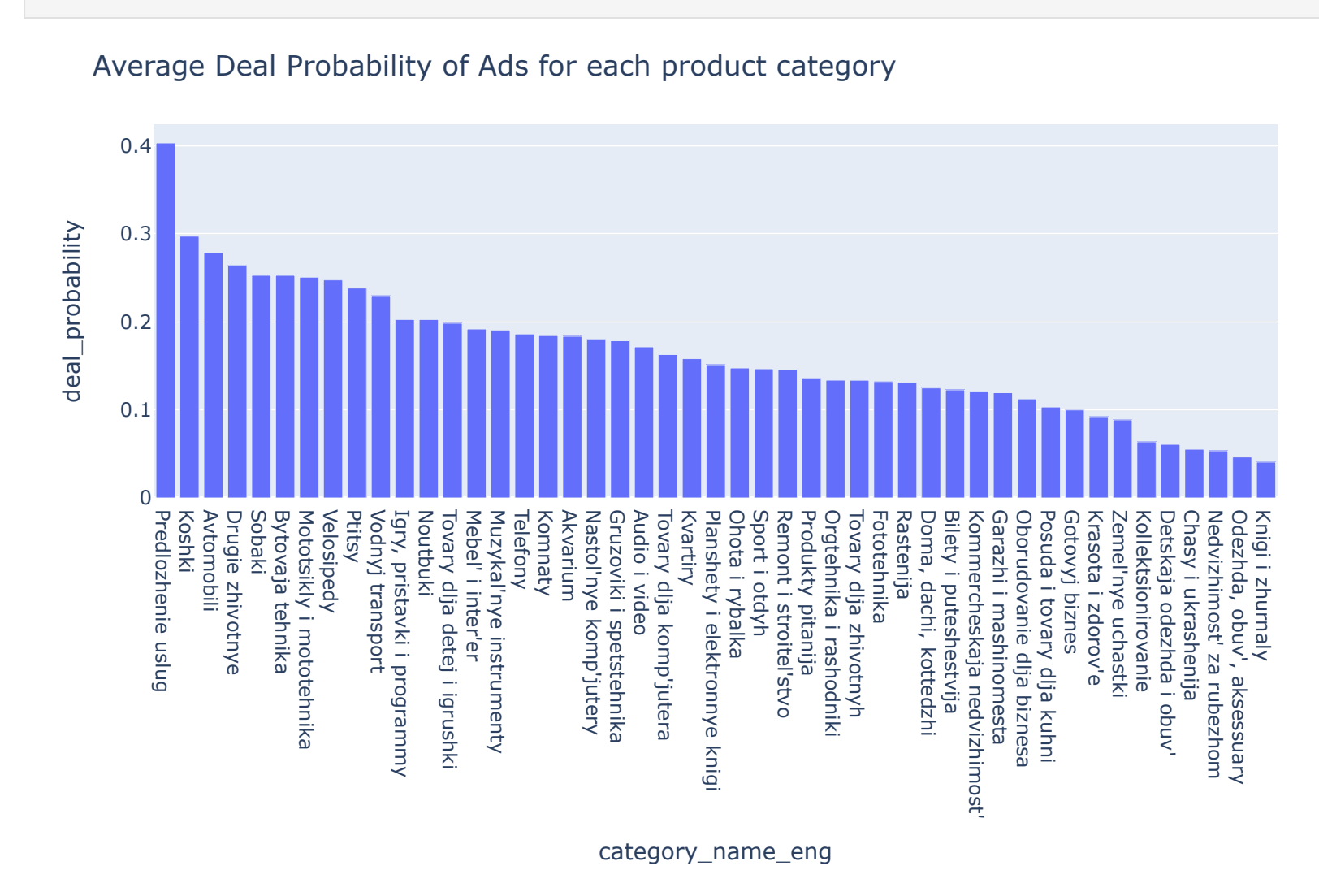
In [4]:
```
user_group=train_df[['user_type','deal_probability']].groupby('user_type',as_index=False).agg('mean')
fig=px.bar(user_group,x='user_type',y='deal_probability')
fig.update_layout(width=500,height=500,title='Deal Probability of Ads from each user_type')
fig.show()
```



Key finding(s):

- Ads from user type 'Shop' have the lowest probability to get a deal.
- Ads from user type 'Private' and 'Company' have the comparatively much higher deal probability than that of 'Shop'.
- This shows that modelling 'user_type' to predict 'deal_probability' can be beneficial.

In [5]:
```
cat_group=train_df[['category_name_eng','deal_probability']].groupby('category_name_eng',as_index=False).agg('m
cat_group=cat_group.sort_values(by='deal_probability',ascending=False,ignore_index=True)
fig=px.bar(cat_group,x='category_name_eng',y='deal_probability')
fig.update_layout(title='Average Deal Probability of Ads for each product category')
fig.show()
```



Key finding(s):

- Average deal probabilities significantly vary by product categories.
- This shows that modelling 'category_name' to predict 'deal_probability' can be beneficial.

Distribution of deal probabilities

In [6]:
```
def prob_class(x):
    if x >=0.5:
        return '>=0.5'
    if x==0.0:
        return '0'
    else:
        return "<0.5"


train_df['prob_class'] = train_df['deal_probability'].apply(prob_class)
temp = train_df['prob_class'].value_counts()
labels = temp.index
sizes = (temp / temp.sum())*100
trace = go.Pie(labels=labels, values=sizes, hoverinfo='label+percent')
layout = go.Layout(title='Distribution of deal probabilities')
data = [trace]
fig = go.Figure(data=data, layout=layout)

fig.show()
del train_df['prob_class']
```

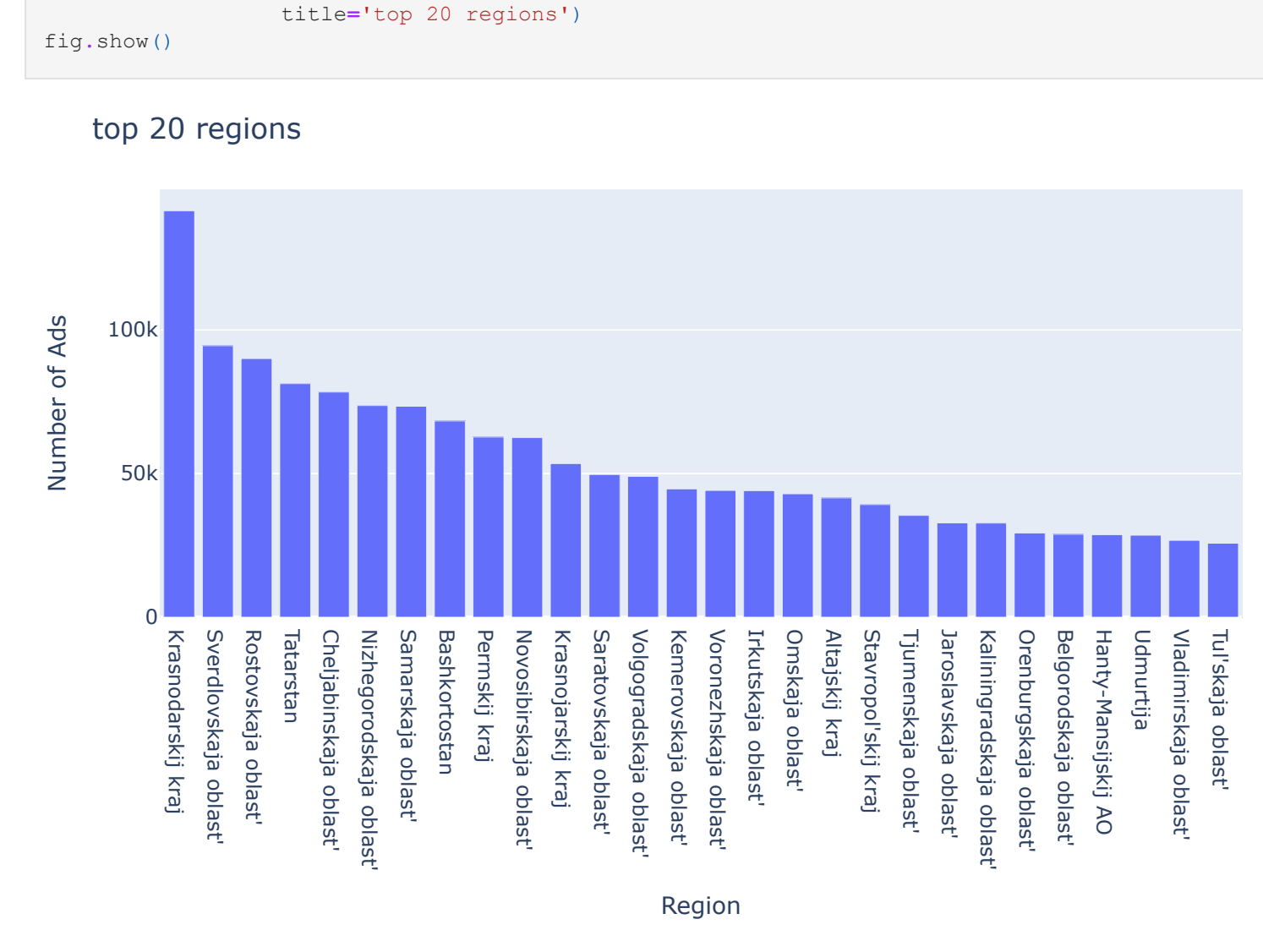
Key finding(s):

- More than 65% of ads have zero deal probability.
- More than 11% of ads have less than 0.5 deal probability.
- Only 20.25% of ads have more than 0.5 deal probability.

Top 20 regions as per number of ads posted

In [7]:
```
temp=train_df['region_eng'].value_counts()

fig = px.bar(
    x = temp.index,
    y = temp.values,orientation='v'
)
fig.update_layout(
                  yaxis_title='Number of Ads',
                  xaxis_title='Region',height=500,width=800,
                  title='top 20 regions')
fig.show()
```



Key finding(s):

- Number of ads posted drastically vary region to region.

Average deal probability for ads with and without mentioned price.

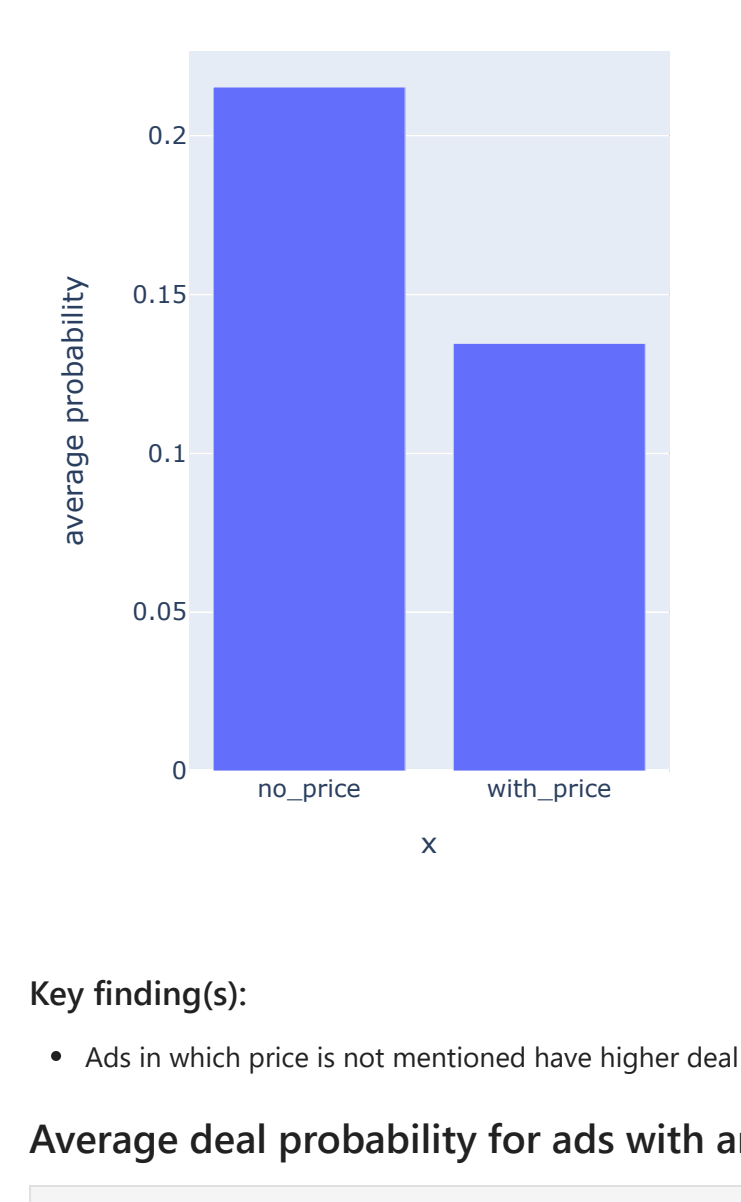
In [8]:
```
no_price=train_df[train_df['price'].isnull()==True]['deal_probability'].mean()

with_price=train_df[train_df['price'].isnull()==False]['deal_probability'].mean()

fig=px.bar(x=['no_price','with_price'],
           y=[no_price,with_price])

fig.update_layout(width=400,height=500,
                  yaxis_title='average probability')

fig.show()
```



Key finding(s):

- Ads in which price is not mentioned have higher deal probability.

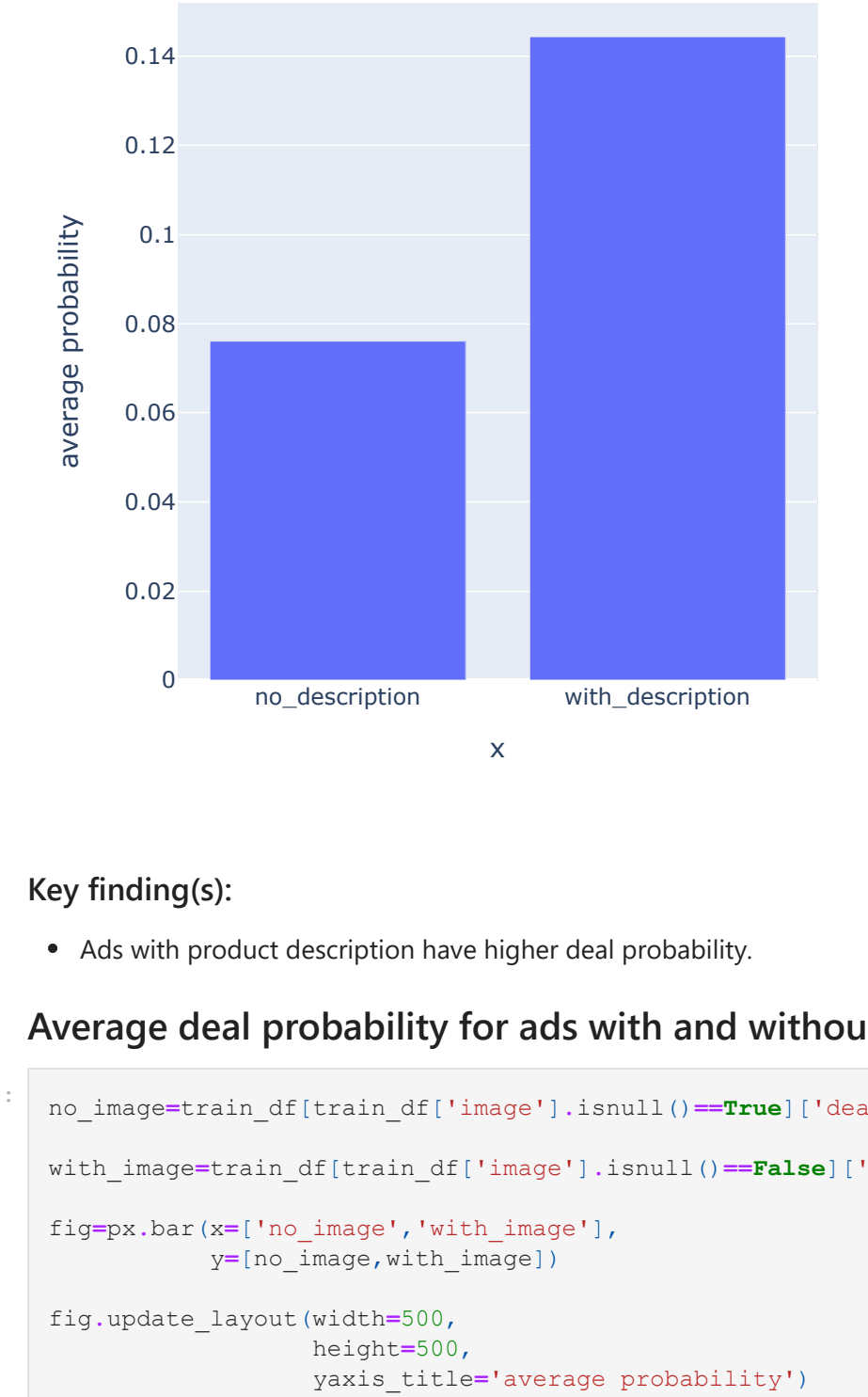
Average deal probability for ads with and without description.

In [9]:
```
no_des=train_df[train_df['description_eng']=='nan']['deal_probability'].mean()

with_des=train_df[train_df['description_eng']!='nan']['deal_probability'].mean()

fig=px.bar(x=['no_description','with_description'],
           y=[no_des,with_des])
fig.update_layout(width=500,
                  height=500,
                  yaxis_title='average probability')

fig.show()
```



Key finding(s):

- Ads with product description have higher deal probability.

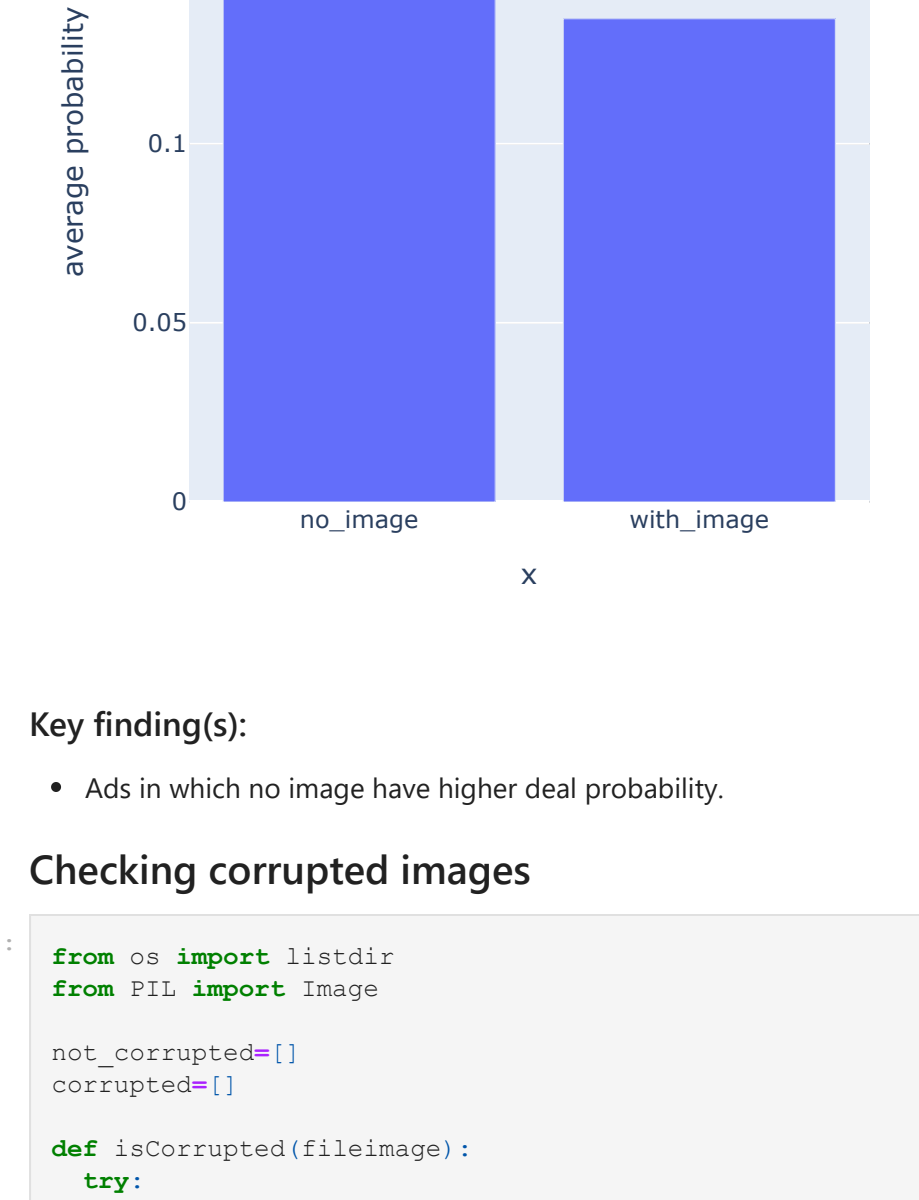
Average deal probability for ads with and without product image.

In [10]:
```
no_image=train_df[train_df['image'].isnull()==True]['deal_probability'].mean()

with_image=train_df[train_df['image'].isnull()==False]['deal_probability'].mean()

fig=px.bar(x=['no_image','with_image'],
           y=[no_image,with_image])

fig.update_layout(width=500,
                  height=500,
                  yaxis_title='average probability')
fig.show()
```



Key finding(s):

- Ads in which no image have higher deal probability.

Checking corrupted images

In [ ]:
```
from os import listdir
from PIL import Image

not_corrupted=[]
corrupted=[]

def isCorrupted(fileimage):
  try:
    with Image.open(fileimage+'.jpg') as img:
      img.verify() # verify that it is, in fact an image
    return False
  except Exception as e:
    print(e)
    return True

img_list=train_df['image'].dropna()
path='/content/images/data/competition_files/train_jpg/'
for filename in img_list:
  if isCorrupted(path+filename)==False:
    not_corrupted.append(filename)

  if isCorrupted(path+filename)==True:
    corrupted.append(filename)
```

In [12]:
```
print('number of corrupted images: ',len(corrupted))
```

```
number of corrupted images: 4
```