Import Liabaries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import random
import gcsfs
import sklearn.metrics as metrics
import datetime
import dask.dataframe as dd
from IPython.display import Markdown as md
from google.cloud import storage
from google.oauth2 import service_account
import json
import warnings
import pickle
%matplotlib inline
```

Ignore Warnings

In [2]:
```
warnings.filterwarnings('ignore')
```

Import the best model from GCP

In [3]:
```
credentials = {
  "type": "service_account",
  "project_id": "phonic-monolith-345108",
  "private_key_id": "09e8b01c72c4bf6f11b35998df3d41d727fb33c8",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDJyfcTdIRQuEo2\nIqWIx4U9Ari+RbF/HrZz+HBYVwL9+mpIvgOySYlKcSdBljhx8dC4vP2YbKhbdpNp\ndZEwRy7vp35wsN2WWHx+IAeiMgOFyDSXnl3Pcl4IdZBmcWwjbQ
  "client_email": "danielevko@phonic-monolith-345108.iam.gserviceaccount.com",
  "client_id": "112729790674474565098",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/danielevko%40phonic-monolith-345108.iam.gserviceaccount.com"
}
```

In [4]:
```
gcp_json_credentials_dict = json.loads(json.dumps(credentials))
creds = service_account.Credentials.from_service_account_info(gcp_json_credentials_dict)
client = storage.Client(project=gcp_json_credentials_dict['project_id'], credentials=creds)
```

In [5]:
```
bucket = client.bucket('final_project_leads')
```

In [6]:
```
blob_model=''
for blob in client.list_blobs('final_project_leads',prefix='danielev'):
    if 'best_model' in str(blob):
        print(blob)
        blob_model=blob
        break
```

```
<Blob: final_project_leads, danielev/best_model_random_forest, 1653483147209057>
```

In [7]:
```
pickle_data = blob_model.download_as_string()
```

In [8]:
```
model = pickle.loads(pickle_data)
```

Import Data from GCP

In [9]:
```
def import_from_gcp(file_name, bucket_name):
    PROJECT_NAME = 'final-project-lead-me'

    URL = "gs://"

    # Creating a pythonic file-system interface to Google Cloud Storage.
    fs = gcsfs.GCSFileSystem(project=PROJECT_NAME)
    for i in fs.ls(bucket_name):
        if file_name in i:
            print(i)
            return pd.read_csv(URL + i, storage_options={"token": credentials}, encoding="ISO-8859-1")
    return print("File not found")
```

In [10]:
```
df_companies = import_from_gcp("Companies Data", "final_project_leads")
df_cars = import_from_gcp("vehicles4", "final_project_leads")
```

```
final_project_leads/Companies Data.csv
final_project_leads/vehicles4.csv
```

Upload a new leads file from the client

In [11]:
```
df_leads = import_from_gcp("sales_leads_DO_NOT_DELETE_PART3", "final_project_leads/danielev")
```

```
final_project_leads/danielev/sales_leads_DO_NOT_DELETE_PART3.csv
```

In [12]:
```
df_leads.head()
```

Out[12]:

	id_lead	id	is_buisness	first_name	last_name	email	gender	year_of_birth	country	address	company_name	platform	department	car_type	car_model	creation_date	creation_time	car_year	rental_period
0	101001	57659904	False	Stephanie	Axten	saxten0@psu.edu	Female	1961	United Kingdom	NaN	Graphic Packaging Holding	Facebook	Research and development	saturn	ion 3	13/05/2018	20	2007	22/06/2018
1	101002	73735325	False	David	Nottingham	dnottingham1@miitbeian.gov.cn	Male	1952	Germany	0586 Blaine Way	WESCO International	Google	Research and development	volkswagen	jetta	04/10/2019	15	2015	03/12/2019
2	101003	56091361	True	Angus	Grigaut	agrigaut2@moonfruit.com	Male	1988	Israel	NaN	Alphabet	Facebook	Human resources	acura	nsx	22/04/2019	10	2017	30/07/2020
3	101004	20700232	False	Angus	Striker	astriker3@amazon.co.uk	Male	1938	Italy	82022 Evergreen Center	Owens & Minor	Website	Engineering	buick	enclave	24/02/2018	4	2010	15/08/2018
4	101005	82047750	False	Neil	Emmott	nemmott4@smugmug.com	Male	1972	United Kingdom	6500 Fair Oaks Point	Huntsman	Google	Engineering	chrysler	town & country	04/12/2019	1	2012	04/03/2020

Data Pre-processing

Data Cleaning

Checking for duplication values in df_leads

In [13]:
```
#checking duplicates id leads
if (sum(df_leads.duplicated(subset = 'id_lead')) == 0):
    print("No Duplication in id_lead")
else:
    #Number of duplications in id_lead
    dups = len(df_leads['id_lead'])-len(df_leads['id_lead'].drop_duplicates())
    print("{} Duplication in id_lead".format(dups))
    df_leads.drop_duplicates(subset=['id_lead'],keep='first',inplace=True)
    #checking after delete duplicates id leads
    sum(df_leads.duplicated(subset = 'id_lead')) == 0
    # No duplicate values
```

```
No Duplication in id_lead
```

In [14]:
```
#checking duplicates id's
if (sum(df_leads.duplicated(subset = 'id')) == 0):
    print("No Duplication in id")
else:
    #Number of duplications in id_lead
    dups = len(df_leads['id'])-len(df_leads['id'].drop_duplicates())
    print("{} Duplication in id".format(dups))
    df_leads.drop_duplicates(subset=['id'],keep='first',inplace=True)
    #checking after delete duplicates id leads
    sum(df_leads.duplicated(subset = 'id')) == 0
```

```
No Duplication in id
```

Calculate Age

In [15]:
```
def calculate_age(born):
    today = datetime.date.today()
    return today.year - born

df_leads['age'] = df_leads['year_of_birth'].apply(calculate_age)
```

Data Cleaning

In [16]:
```
df_companies=df_companies[df_companies['Market Cap']!='-']
df_companies=df_companies[df_companies['Market Cap'].notna()]
df_companies=df_companies[df_companies['company_name'].map(lambda x: x.isascii())]
df_companies.shape
```

Out[16]:
```
(947, 19)
```

In [17]:
```
df_companies=df_companies.sort_values('profit',ascending=False)
print(df_companies.shape)
```

```
(947, 19)
```

Cleaning the cars Dataset

In [18]:
```
md("Cleaning cars whose year is than %i"%(df_leads['car_year'].min()))
```

Out[18]: Cleaning cars whose year is than 1998

In [19]:
```
df_cars=df_cars[df_cars['year']>=df_leads['car_year'].min()]
```

In [20]:
```
df_cars.shape
```

Out[20]:
```
(341327, 7)
```

In [21]:
```
if len(df_leads[df_leads['car_year']<1997])==0:
    print(True)
else:
    False
```

```
True
```

Cleaning cars that do not include the maunfactuer from df_leads

In [22]:
```
manufacturers_to_drop=np.unique(df_cars['car'])[~np.in1d(np.unique(df_cars['car']),np.unique(df_leads['car_type']))].tolist()
manufacturers_to_drop
```

Out[22]:
```
['aston-martin', 'jaguar', 'land rover', 'tesla']
```

In [23]:
```
df_cars=df_cars[df_cars.car.isin(manufacturers_to_drop) == False]
```

Cleaning cars that do not include the same model from df_leads

In [24]:
```
models_to_drop=np.asarray(list(set(df_cars['model'])))[~np.in1d(np.asarray(list(set(df_cars['model']))),np.unique(df_leads['car_model']))]
models_to_drop
```

Out[24]:
```
array(['nan',
       'silverado 3500 diesel trucks 6.6l duramax lbz with allison 1000 6 speed automatic',
       'c-5500', ..., 'prius hybrid cab', '718', '4runner sport'],
      dtype='<U192')
```

In [25]:
```
df_cars=df_cars[df_cars.model.isin(models_to_drop) == False]
```

In [26]:
```
df_cars.shape
```

Out[26]:
```
(95018, 7)
```

df_leads: Data Transformation

In this part we are going to match prices of cars in our df_leads to df_cars

Create new df that is group by car_type and year

In [27]:
```
df_mean_car_by_model_and_year=df_cars.groupby(['car','model','year']).mean()
df_mean_car_by_model_and_year['price']=df_mean_car_by_model_and_year['price'].map(lambda x: round(x,2))
df_mean_car_by_model_and_year
```

Out[27]:

car	model	year	id_car	price
acura	ilx	2013	7.311279e+09	10949.53
		2014	7.313938e+09	14309.54
		2015	7.311765e+09	13216.33
		2016	7.313127e+09	15939.81
		2017	7.312173e+09	15769.86
...	...	...	...	...
volvo	xc90	2017	7.306397e+09	36431.07
		2018	7.309871e+09	36658.73
		2019	7.314384e+09	49313.17
		2020	7.309088e+09	56328.33
		2021	7.306675e+09	61999.00

3901 rows × 2 columns

Calculate the average car price based on the type and the year of the car, if there is no year information, calculate the average based on the type of car

In [28]:
```
temp_prices=[]
for index,row in df_leads.iterrows():
    #If the car_type is in df_cars it will enter
    try:
        #Creating a new Dataframe which grouped by the car_type
        new_df=df_mean_car_by_model_and_year.loc[df_mean_car_by_model_and_year.index].loc[row['car_type']].reset_index()
        #If the car_model is in df_cars, will select only the prices of the model
        if row['car_model'] in new_df['model'].values:
            new_df=new_df[new_df['model']==row['car_model']]
            #if there is info of model and year, will get the price of the model and year
            if row['car_year'] in new_df['year'].values:
                temp=new_df[(new_df['model']==row['car_model']) & (new_df['year']==row['car_year'])]['price'].values[0]
                temp=round(temp,2)
                temp_prices.append(temp)
            else:
                #if not, will give the minimum year price.
                temp_prices.append(new_df[new_df['price']==min(new_df['price'])]['price'].values[0])
        else:
            #if there is no info of car_model, will give the car_price by year
            new_df=new_df.groupby(['year']).mean()
            if row['car_year'] in new_df.index:
                temp_prices.append(new_df.loc[row['car_year']]['price'])
            else:
                temp_prices.append(new_df.loc[min(new_df.index)]['price'])
    except:
        #If the car type is not located in df_cars it will calculate price by the car year
        new_df=df_cars.groupby(['year']).mean()
        if row['car_year'] in new_df.index:
            temp_prices.append(new_df.loc[row['car_year']]['price'])
        else:
            #if the year is not located it will calculate by the minimum year
            temp_prices.append(new_df.loc[min(new_df.index)]['price'])
```

In [29]:
```
df_leads['car_price']=temp_prices
```

Convert the dates data to date_type in df_leads

Convert to datetime to proper datatype

In [30]:
```
df_leads['rental_period']=pd.to_datetime(df_leads['rental_period'],format='%d/%m/%Y')
df_leads['creation_date']=pd.to_datetime(df_leads['creation_date'],format='%d/%m/%Y')
df_leads[['rental_period','creation_date']].head()
```

Out[30]:

	rental_period	creation_date
0	2018-06-22	2018-05-13
1	2019-12-03	2019-10-04
2	2020-07-30	2019-04-22
3	2018-08-15	2018-02-24
4	2020-03-04	2019-12-04

Create new column based on creation_date and rental_period that called the diff of them, called 'desirable_rental_days'

In [31]:
```
df_leads['desirable_rental_days']=df_leads['rental_period']-df_leads['creation_date']
df_leads['desirable_rental_days']=df_leads['desirable_rental_days'].dt.days
```

Verify that the users filled out the rental period correctly

In [32]:
```
if len(df_leads[df_leads['desirable_rental_days']<0])>=1:
    print(False)
else:
    print(True)
```

```
True
```

Data Merge between df_leads and df_companies

verify all the data in columns Market Cap and Profit is numeric

In [33]:
```
try:
    df_companies['Market Cap']=pd.to_numeric(df_companies['Market Cap'])
    df_companies['profit']=pd.to_numeric(df_companies['profit'])
    print("Succeed")
except:
    print("Error occured in converting Profit and Market Cap")
```

```
Succeed
```

Create a copy of df_companies

In [34]:
```
df_company_to_merge=df_companies.copy()
```

Drop unwanted columns in df_company_to_merge

In [35]:
```
df_company_to_merge.drop(['id_company','num. of employees','profitable','rank_change','rank','city','state','newcomer','ceo_founder','ceo_woman','prev_rank','CEO','Website','Ticker','sector'],inplace=True,axis=1)
```

Make a copy of df_leads before merge

In [36]:
```
df_leads_for_analysis=df_leads.copy()
df_leads_for_analysis.sort_values(by="id_lead",inplace=True)
```

Merge df_company_to_merge and df_leads

In [37]:
```
df_leads_for_analysis=df_leads.merge(df_company_to_merge,on='company_name',sort=False)
```

Sort id_lead Dataframe by id (To return to the original order)

In [38]:
```
df_leads_for_analysis.sort_values(by="id_lead",inplace=True)
```

This is the Dataset the client will get to preform sales

In [39]:
```
df_leads_app = df_leads_for_analysis.copy()
```

Drop unwanted columns

In [40]:
```
df_leads_for_analysis.drop(['id','id_lead','first_name','last_name','email','year_of_birth','country','address','creation_date','rental_period','car_type','company_name','car_model'],axis=1,inplace=True)
```

In [41]:
```
df_leads_for_analysis.head()
```

Out[41]:

	is_buisness	gender	platform	department	creation_time	car_year	age	car_price	desirable_rental_days	revenue	profit	Market Cap
0	False	Female	Facebook	Research and development	20	2007	61	4749.50	40	6160.1	206.8	3542.0
4	False	Male	Google	Research and development	15	2015	70	9360.78	60	8358.9	223.4	4294.0
18	True	Male	Facebook	Human resources	10	2017	34	127971.50	465	161857.0	34343.0	1274139.0
20	False	Male	Website	Engineering	4	2010	84	8577.74	172	9650.0	-62.4	576.9
22	False	Male	Google	Engineering	1	2012	50	8604.47	91	8342.0	562.0	3217.2

Converting Categorical Features

Convert the 'is_buisness' boolean column to String type

In [42]:
```
df_leads_for_analysis['is_buisness'] = df_leads_for_analysis['is_buisness'].map({True: 'True', False: 'False'})  # Replace boolean to string
```

Create a list of categorical columns

In [43]:
```
categorical_columns = [i for i in range(len(np.array(df_leads_for_analysis.dtypes))) if np.array(df_leads_for_analysis.dtypes)[i] not in [np.dtype('float64'),np.dtype('int64')]]
```

In [44]:
```
df_dummies=pd.get_dummies(df_leads_for_analysis[df_leads_for_analysis.columns[categorical_columns]],drop_first=True)
```

Drop the Categorical columns from df_leads_for_analysis

In [45]:
```
df_leads_for_analysis.drop(df_leads_for_analysis.columns[categorical_columns],axis=1,inplace=True)
```

Concat all the df_dummies

In [46]:
```
df_leads_for_analysis = pd.concat([df_leads_for_analysis,df_dummies,],axis=1)
```

Make predictions on df_leads_for_analysis

In [47]:
```
predictions = model.predict(df_leads_for_analysis)
```

Results of predictions

In [48]:
```
unique, counts = np.unique(predictions, return_counts=True)
print (np.asarray((unique, counts)).T)
```

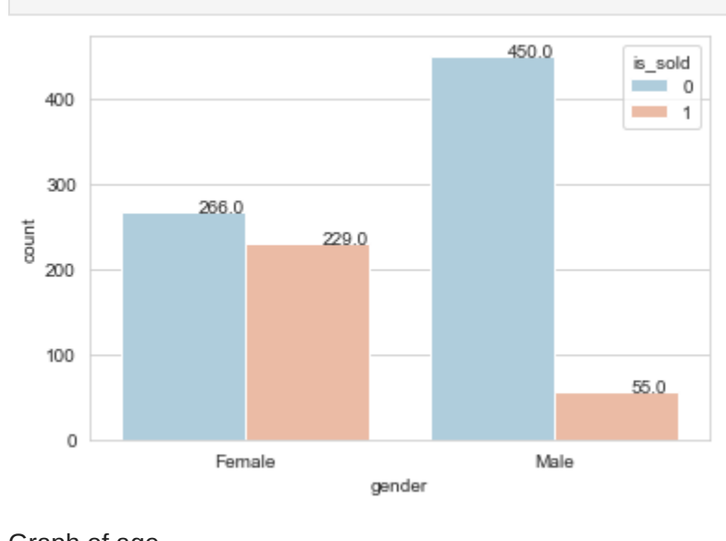
```
[[  0 716]
 [  1 284]]
```

In [49]:
```
df_leads_app['is_sold'] = predictions
```

Analyizing the Model results

Male vs. Female is sold

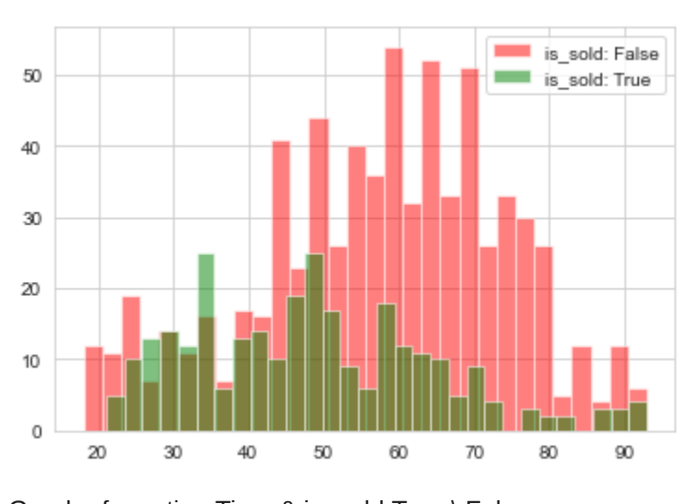
In [50]:
```
sns.set_style('whitegrid')
sns_plot_female_vs_male_buisness=sns.countplot(x='gender',hue='is_sold',data=df_leads_app,palette='RdBu_r')
sns_plot_female_vs_male_buisness.figure.savefig("Male vs. Female is_Sold.jpeg")
for p in sns_plot_female_vs_male_buisness.patches:
        sns_plot_female_vs_male_buisness.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
plt.show()
```



Graph of age

In [51]:
```
sns_plot_age_vs_not_sold=df_leads_app[df_leads_app['is_sold']==0]['age'].hist(bins=30,color='red',alpha=0.5)
sns_plot_age_vs_sold=df_leads_app[df_leads_app['is_sold']==1]['age'].hist(bins=30,color='green',alpha=0.5)
plt.legend(labels=["is_sold: False","is_sold: True"])
```

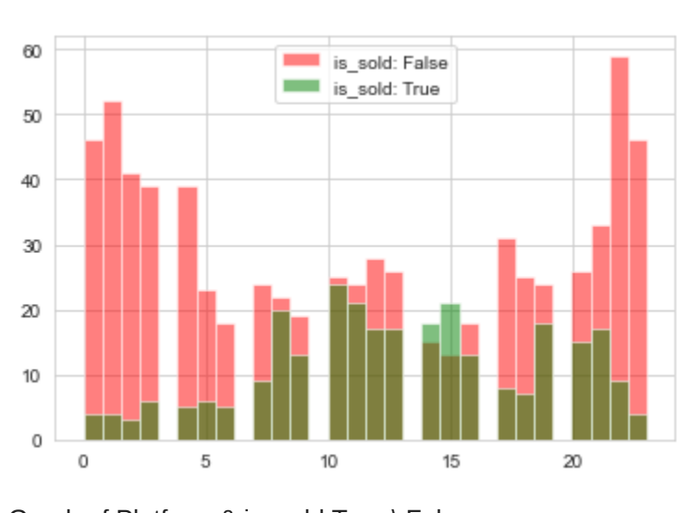
Out[51]:
```
<matplotlib.legend.Legend at 0x1ea3259b070>
```



Graph of creation Time & is_sold True \ False

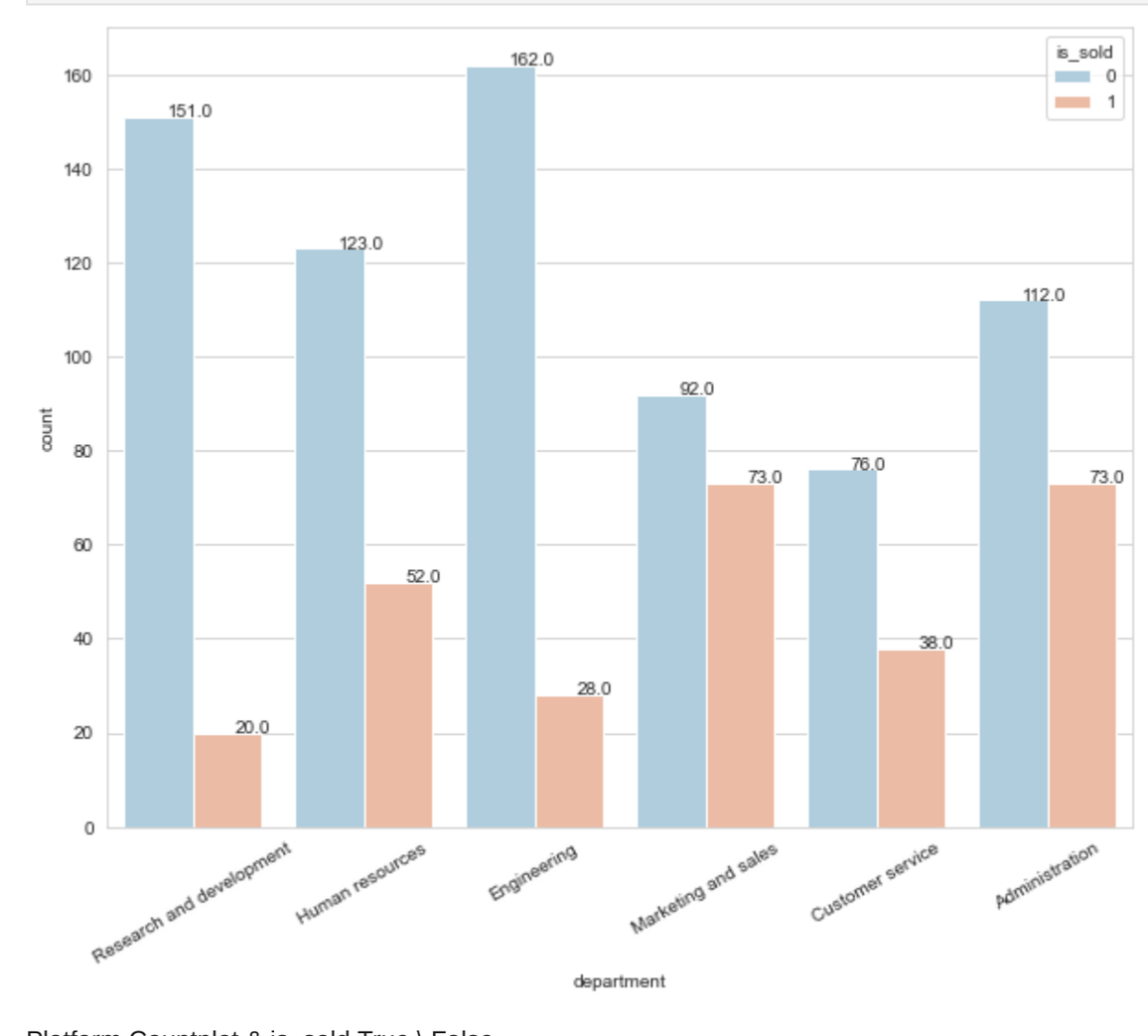
In [52]:
```
sns_plot_creation_time=df_leads_app[df_leads_app['is_sold']==False]['creation_time'].hist(bins=30,color='red',alpha=0.5)
sns_plot_creation_time=df_leads_app[df_leads_app['is_sold']==True]['creation_time'].hist(bins=30,color='green',alpha=0.5)
plt.legend(labels=["is_sold: False","is_sold: True"])
```

Out[52]:
```
<matplotlib.legend.Legend at 0x1ea335ae8e0>
```



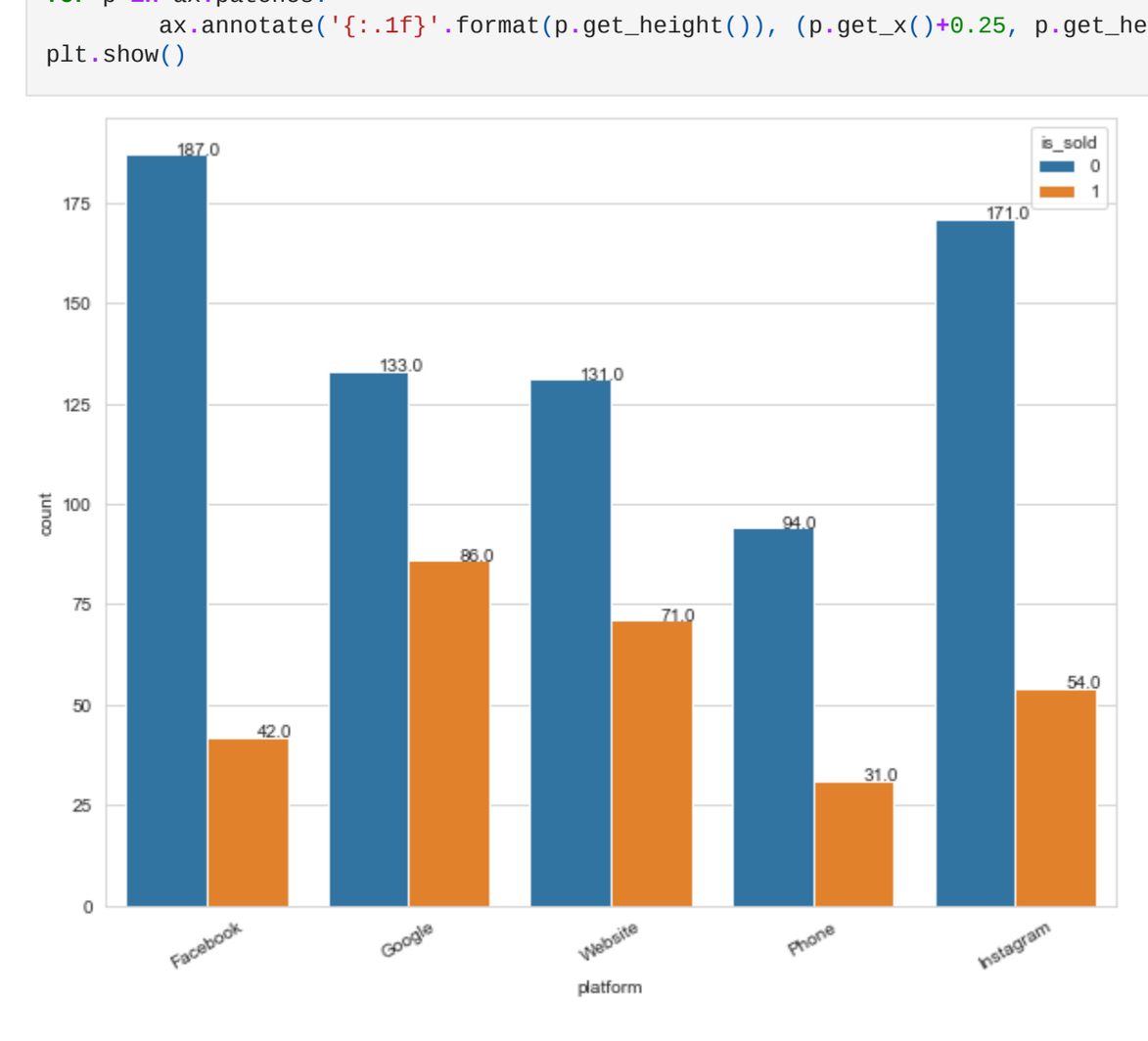
Graph of Platform & is_sold True \ False

In [53]:
```
sns.set_style('whitegrid')
plt.figure(figsize = (10,8))
ax = sns.countplot(x='department',data=df_leads_app,palette='RdBu_r',hue='is_sold')
ax.set_xticklabels(ax.get_xticklabels(),rotation = 30)
for p in ax.patches:
        ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
plt.show()
```



Platform Countplot & is_sold True \ False

In [54]:
```
sns.set_style('whitegrid')
plt.figure(figsize = (10,8))
ax = sns.countplot(x='platform',data=df_leads_app,hue='is_sold')
ax.set_xticklabels(ax.get_xticklabels(),rotation = 30)
for p in ax.patches:
        ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+0.25, p.get_height()+0.01))
plt.show()
```



Summarize Results

In [55]:
```
result_data=df_leads_app.copy()
result_data['Result'] = result_data['is_sold'].map({0:'No', 1:'Yes'})
# Order the cluster
result_data['Result'] = result_data['is_sold'].astype('category')
```

In [56]:
```
result_data.rename(columns = {'is_sold':'Total'}, inplace = True)
df_groupby_segmant=result_data.groupby('Result').agg(
    {
        'Total':'count',
        'is_buisness': lambda x: x.value_counts().index[0],
        'gender': lambda x: x.value_counts().index[0],
        'department': lambda x: x.value_counts().index[0],
        'car_year': 'median',
        'platform': lambda x: x.value_counts().index[0],
        'age': np.mean,
        'car_price': 'mean',
        'desirable_rental_days': 'median',
        'Market Cap':'mean',
        'revenue':'mean',
        'profit':'mean',
        'Market Cap':'mean'
    }
).reset_index()
df_groupby_segmant
```

Out[56]:

	Result	Total	is_buisness	gender	department	car_year	platform	age	car_price	desirable_rental_days	Market Cap	revenue	profit
0	0	716	False	Male	Engineering	2013.0	Facebook	56.923184	11590.724120	248.5	10894.927793	13771.097765	325.449022
1	1	284	True	Female	Marketing and sales	2016.0	Google	48.151408	35231.386514	517.5	106025.614085	40753.657746	4852.113028

Transform the DF to dask.dataframe and upload to the Cloud

Sort the file by is_sold

In [57]:
```
df_leads_app.sort_values(by="is_sold",inplace=True,ascending=False)
```

Upload the file with today's date

In [58]:
```
def today_date(i):
    today = datetime.date.today()
    return str(today)
```

In [59]:
```
ddf = dd.from_pandas(df_leads_app,npartitions=1,sort=False)
ddf.to_csv('gs://final_project_leads/danielev/leads_after_prediction-*.csv', index=False, sep=',', header=True,name_function=today_date,
                              storage_options={'token':credentials})
```

Out[59]:
```
['final_project_leads/danielev/leads_after_prediction-2022-05-25.csv']
```