Project name :- Freelance Platform Project

Type:-Supervised Machine Learning (Classification model)

Quck Introduction :- The freelance industry has experienced significant growth in recent years, with online platforms connecting freelancers and clients across various industries. This project aimed to perform an exploratory data analysis on a dataset from a freelance platform to gain insights into the dynamics and trends of the platform. By analyzing the data, we aimed to understand the characteristics of freelancers, client preferences, and job postings on the platform.Simultaneously we will use different algorithms to see prediction on the Type(Fixed or Hourly)

Import libraries

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

Load the data

In [2]:
```
df=pd.read_csv('FreelancePlatformProject.csv')

df.head() #Reading the top 5 data
```

Out[2]:

	Title	Category Name	Experience	Sub Category Name	Currency	Budget	Location	Freelancer Preferred From	Type	Date Posted	Description	Duration	Client Registration Date	Client City	Client Country	Client Currency	Client Job Title
0	Banner images for web desgin websites	Design	Entry ($)	Graphic Design	EUR	60	remote	ALL	fixed_price	29-04-2023 18:06	We are looking to improve the banner images on...	NaN	03-11-2010	Dublin	Ireland	EUR	PPC Management
1	Make my picture a solid silhouette	Video, Photo & Image	Entry ($)	Image Editing	GBP	20	remote	ALL	fixed_price	29-04-2023 17:40	Hello \n\nI need a quick designer to make 4 pi...	NaN	21-02-2017	London	United Kingdom	GBP	Office manager
2	Bookkeeper needed	Business	Entry ($)	Finance & Accounting	GBP	12	remote	ALL	fixed_price	29-04-2023 17:40	Hi - I need a bookkeeper to assist with bookke...	NaN	09-04-2023	London	United Kingdom	GBP	Paralegal
3	Accountant needed	Business	Entry ($)	Tax Consulting & Advising	GBP	14	remote	ALL	fixed_price	29-04-2023 17:32	Hi - I need an accountant to assist me with un...	NaN	09-04-2023	London	United Kingdom	GBP	Paralegal
4	Guest Post on High DA Website	Digital Marketing	Expert ($$$)	SEO	USD	10000	remote	ALL	fixed_price	29-04-2023 17:09	Hi, I am currently running a project where I w...	NaN	01-07-2016	Mumbai	India	USD	Guest posts buyer

Understanding the data:

Let's examine the data to get a better understanding of its structure and contents.

In [3]:
```
print('Shape of our dataframe is:',df.shape)
```

```
Shape of our dataframe is: (12202, 17)
```

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

#Number of columns are:17
#Budget is the only Numerical columns else are Categorical
#Duration and Client job title columns contains null values
#Type is our dependent variable and other balance features are independent variables
#Output variable is not continuous
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12202 entries, 0 to 12201
Data columns (total 17 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   Title                      12202 non-null  object
 1   Category Name              12202 non-null  object
 2   Experience                 12202 non-null  object
 3   Sub Category Name          12202 non-null  object
 4   Currency                   12202 non-null  object
 5   Budget                     12202 non-null  int64 
 6   Location                   12202 non-null  object
 7   Freelancer Preferred From  12202 non-null  object
 8   Type                       12202 non-null  object
 9   Date Posted                12202 non-null  object
 10  Description                12202 non-null  object
 11  Duration                   1602 non-null   object
 12  Client Registration Date   12202 non-null  object
 13  Client City                12202 non-null  object
 14  Client Country             12202 non-null  object
 15  Client Currency            12202 non-null  object
 16  Client Job Title           4581 non-null   object
dtypes: int64(1), object(16)
memory usage: 1.6+ MB
```

Checking Duplicate Values

In [5]:
```
df.duplicated().sum()

#There are one duplicate values present in our dataframe
```

Out[5]: 1

In [6]:
```
# Drop duplicate rows from the DataFrame

df.drop_duplicates(inplace=True)
```

In [7]:
```
df.duplicated().sum()

#Successfully removed the duplicate values
```

Out[7]: 0

Handling missing data

In [8]:
```
#Check if there are any missing values

df.isnull().sum()

#Duration column has null values=10599
#Client Job title has null values=7620
```

Out[8]:
```
Title                          0
Category Name                  0
Experience                     0
Sub Category Name              0
Currency                       0
Budget                         0
Location                       0
Freelancer Preferred From      0
Type                           0
Date Posted                    0
Description                    0
Duration                   10599
Client Registration Date       0
Client City                    0
Client Country                 0
Client Currency                0
Client Job Title            7620
dtype: int64
```

In [9]:
```
#Checking how much % of missing values present in dataset

df.isnull().sum() / len(df) * 100
```

Out[9]:
```
Title                        0.000000
Category Name                0.000000
Experience                   0.000000
Sub Category Name            0.000000
Currency                     0.000000
Budget                       0.000000
Location                     0.000000
Freelancer Preferred From    0.000000
Type                         0.000000
Date Posted                  0.000000
Description                  0.000000
Duration                    86.869929
Client Registration Date     0.000000
Client City                  0.000000
Client Country               0.000000
Client Currency              0.000000
Client Job Title            62.453897
dtype: float64
```

In [10]:
```
#Duration column has 86% null values & Client job title has 62% null values. Dropping both columns

df.dropna(axis=1,inplace=True)
```

In [11]:
```
#Checking the shape after dropping 2 columns

print('the shape of dataset is:-',df.shape)
```

```
the shape of dataset is:- (12201, 15)
```

Data Cleaning

In [12]:
```
df['Experience'].head()
```

Out[12]:
```
0       Entry ($)
1       Entry ($)
2       Entry ($)
3       Entry ($)
4    Expert ($$$)
Name: Experience, dtype: object
```

In [13]:
```
#Removing ($),($$),($$$) symbol and parenthesis from Experience feature by replace function

df['Experience'] = df['Experience'].str.replace('$','').str.replace('(','').str.replace(')','')
df['Experience'].head()
```

Out[13]:
```
0     Entry
1     Entry
2     Entry
3     Entry
4    Expert
Name: Experience, dtype: object
```

In [14]:
```
#Converting all Budget values in usd currency

conversion_rates = {'EUR': 1.07, 'GBP': 1.24, 'USD': 1}

df['Budget_usd'] = df['Currency'].map(conversion_rates) * df['Budget']
df.head(3)
```

Out[14]:

	Title	Category Name	Experience	Sub Category Name	Currency	Budget	Location	Freelancer Preferred From	Type	Date Posted	Description	Client Registration Date	Client City	Client Country	Client Currency	Budget_usd
0	Banner images for web desgin websites	Design	Entry	Graphic Design	EUR	60	remote	ALL	fixed_price	29-04-2023 18:06	We are looking to improve the banner images on...	03-11-2010	Dublin	Ireland	EUR	64.20
1	Make my picture a solid silhouette	Video, Photo & Image	Entry	Image Editing	GBP	20	remote	ALL	fixed_price	29-04-2023 17:40	Hello \n\nI need a quick designer to make 4 pi...	21-02-2017	London	United Kingdom	GBP	24.80
2	Bookkeeper needed	Business	Entry	Finance & Accounting	GBP	12	remote	ALL	fixed_price	29-04-2023 17:40	Hi - I need a bookkeeper to assist with bookke...	09-04-2023	London	United Kingdom	GBP	14.88

In [15]:
```
#We don't need Budget and Currency column as we have created a new column,so dropping Budget

df.drop(['Budget'],axis=1, inplace=True)
```

In [16]:
```
df['Currency'].value_counts()
```

Out[16]:
```
Currency
GBP    8175
USD    3144
EUR     882
Name: count, dtype: int64
```

In [17]:
```
df["Currency"] = "USD" #Assignning USD to all currency
```

In [18]:
```
#Checking the shape after dropping 2 columns

print('The shape of dataset is:-',df.shape)
```

```
The shape of dataset is:- (12201, 15)
```

In [19]:
```
#Exploring the statistical information

df.describe()
```

Out[19]:

	Budget_usd
count	12201.000000
mean	266.237785
std	2282.447586
min	7.440000
25%	37.200000
50%	93.000000
75%	186.000000
max	123998.760000

In [20]:
```
# #Client Registration Date column having object datatype so converting its datatype and extracting it to day/month/year format

df['Client Registration Date'] = pd.to_datetime(df['Client Registration Date'], format='%d-%m-%Y', errors='coerce')
df['Client Registration date'] = df['Client Registration Date'].dt.day
df['Client Registration Month'] = df['Client Registration Date'].dt.month
df['Client Registration Year'] = df['Client Registration Date'].dt.year
```

In [21]:
```
#Date Posted column having object datatype so converting its datatype and extracting it to day/month/year format

df['Date Posted'] = pd.to_datetime(df['Date Posted'], format='%d-%m-%Y %H:%M')
df['Date Posted in Date'] = df['Date Posted'].dt.day
df['Date Posted in Month'] = df['Date Posted'].dt.month
df['Date Posted in Year'] = df['Date Posted'].dt.year
df['Date Posted in Time'] = df['Date Posted'].dt.time
```

In [22]:
```
#Dropping date posted column as we already have done feature extraction

df.drop(['Date Posted','Client Registration Date'],axis=1, inplace=True)
```

In [23]:
```
#Checking the shape after dropping Date Posted column

print('The shape of dataset is:-',df.shape)
```

```
The shape of dataset is:- (12201, 20)
```

Checking skewness

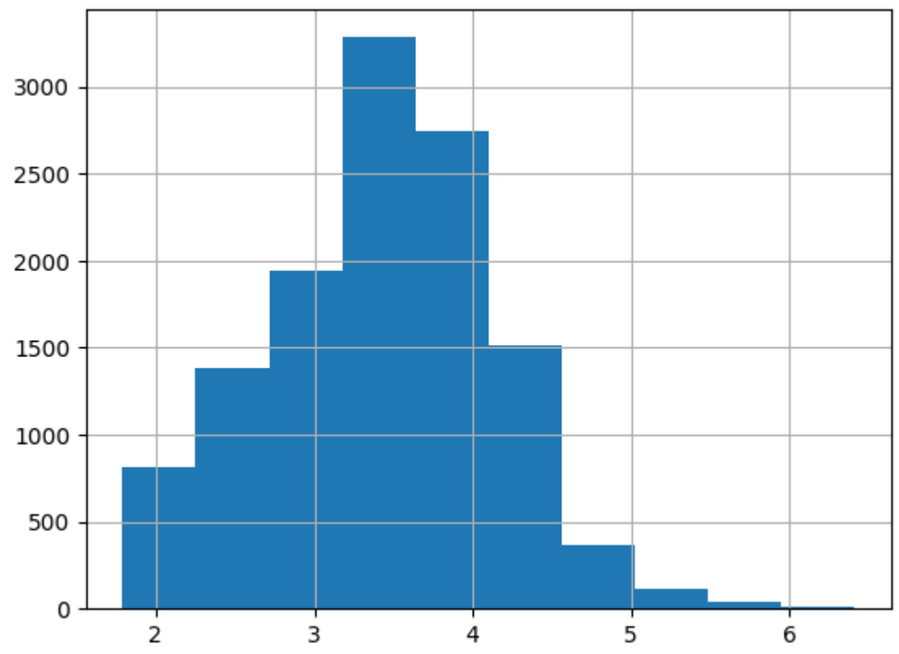
In [24]:
```
df['Budget_usd'].skew()
```

Out[24]:
```
43.9364193416511
```

In [25]:
```
#Box cox transformation

from scipy.stats import boxcox
import numpy as np

after_boxcox = boxcox(df['Budget_usd'])
after_boxcox = pd.Series(after_boxcox[0])
df['Budget_usd'] = after_boxcox

after_boxcox.skew() #We successfully reduced the skewness
```

Out[25]:
```
-0.002291121221937113
```

In [26]:
```
#Checking skewness by visulization, we can see data is normally distributed with bell shape curve

import matplotlib.pyplot as plt
plt.hist(df['Budget_usd'])
plt.grid()

plt.show()
```

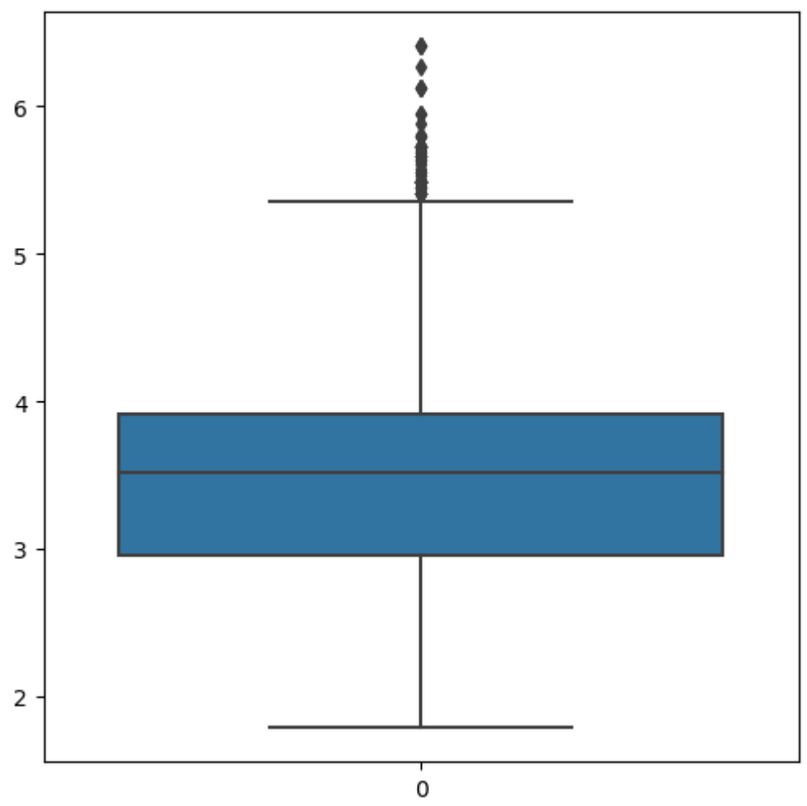


In [27]:
```
#Budget_usd column has null value in one row, so dropping

df.dropna(axis = 0, inplace = True)
```

Handling Outliers

In [28]:
```
#Checking the Outliers by boxplot

plt.figure(figsize=(6,6))

sns.boxplot(df['Budget_usd']) #Need to remove outliers present in the dataframe
```

Out[28]: <Axes: >



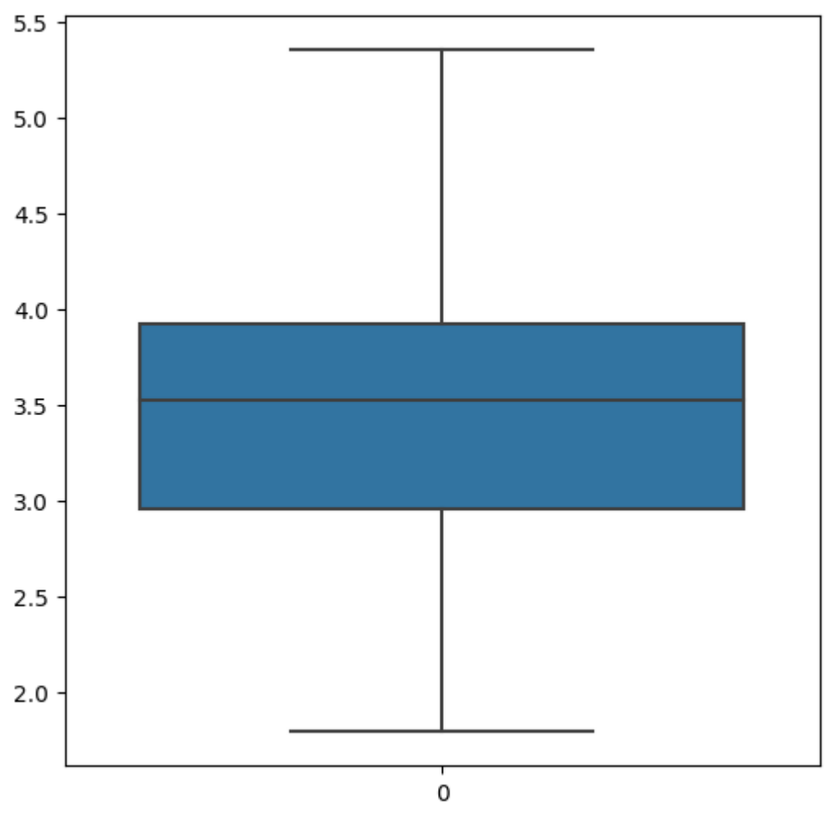
In [29]:
```
#Counting the values that are greater then 5.4

# Convert 'Budget_usd' column to numeric data type
# df['Budget_usd'] = pd.to_numeric(df['Budget_usd'], errors='coerce')

# Count the values greater than 5.4
count = (df['Budget_usd'] > 5.4).sum()
print("Count of values greater than 5.4:", count)
```

```
Count of values greater than 5.4: 63
```

In [30]:
```
#Dropping the outliers from Budget_usd

df= df.drop(df[df['Budget_usd'] > 5.4].index)
```

In [31]:
```
#Checking the Outliers after removing

plt.figure(figsize=(6,6))

sns.boxplot(df['Budget_usd']) #We successfully eliminated all outliers
```

Out[31]: <Axes: >



In [32]:
```
print('The shape of dataset is:-',df.shape)
```

```
The shape of dataset is:- (12137, 20)
```

In [33]:
```
df.dtypes
```

Out[33]:
```
Title                        object
Category Name                object
Experience                   object
Sub Category Name            object
Currency                     object
Location                     object
Freelancer Preferred From    object
Type                         object
Description                  object
Client City                  object
Client Country               object
Client Currency              object
Budget_usd                  float64
Client Registration date      int32
Client Registration Month     int32
Client Registration Year      int32
Date Posted in Date           int32
Date Posted in Month          int32
Date Posted in Year           int32
Date Posted in Time          object
dtype: object
```

In [34]:
```
df.head()
```

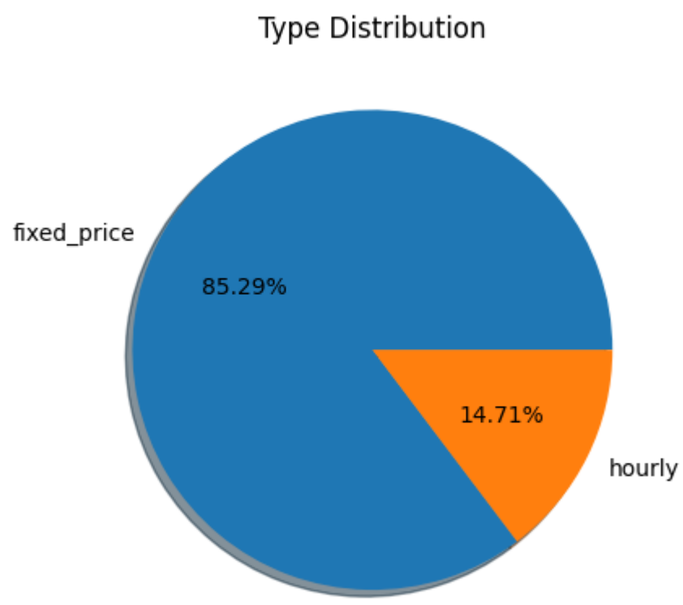
Out[34]:

	Title	Category Name	Experience	Sub Category Name	Currency	Location	Freelancer Preferred From	Type	Description	Client City	Client Country	Client Currency	Budget_usd	Client Registration date	Client Registration Month	Client Registration Year	Date Posted in Date	Date Posted in Month	Date Posted in Year	D Pos in Ti
0	Banner images for web desgin websites	Design	Entry	Graphic Design	USD	remote	ALL	fixed_price	We are looking to improve the banner images on...	Dublin	Ireland	EUR	3.301078	3	11	2010	29	4	2023	18:0(
1	Make my picture a solid silhouette	Video, Photo & Image	Entry	Image Editing	USD	remote	ALL	fixed_price	Hello \n\nI need a quick designer to make 4 pi...	London	United Kingdom	GBP	2.680607	21	2	2017	29	4	2023	17:4(
2	Bookkeeper needed	Business	Entry	Finance & Accounting	USD	remote	ALL	fixed_price	Hi - I need a bookkeeper to assist with bookke...	London	United Kingdom	GBP	2.318016	9	4	2023	29	4	2023	17:4(
3	Accountant needed	Business	Entry	Tax Consulting & Advising	USD	remote	ALL	fixed_price	Hi - I need an accountant to assist me with un...	London	United Kingdom	GBP	2.429707	9	4	2023	29	4	2023	17:32
5	Content Database Project for Travel Company	Technology & Programming	Expert	Databases	USD	remote	ALL	fixed_price	Brief\nThe requirements of this brief is to fi...	Dubai	United Arab Emirates	EUR	4.460790	14	9	2013	29	4	2023	17:0(

Data Analysis

Understanding the Distribution of Type

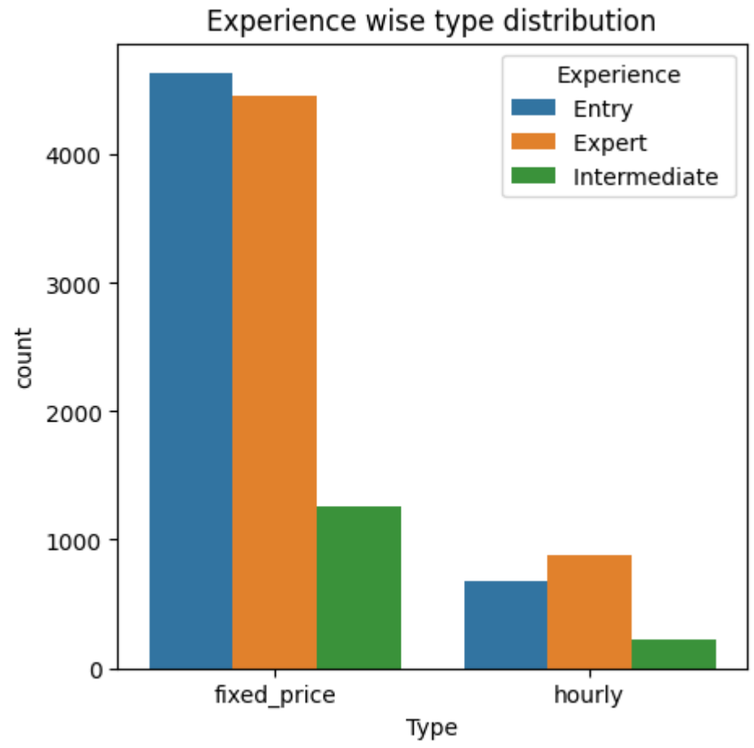
In [35]:
```
category_counts = df['Type'].value_counts()

# Plotting the pie chart
plt.pie(category_counts, labels=category_counts.index, autopct='%0.2f%%', shadow=True)

# Adding a title to the chart
plt.title("Type Distribution")

# Show the plot
plt.show()
```



From above graph we can understand that the Fixed price has the highest and hourly type has the lowest distribution

Understanding the Distribution of Experience with Type

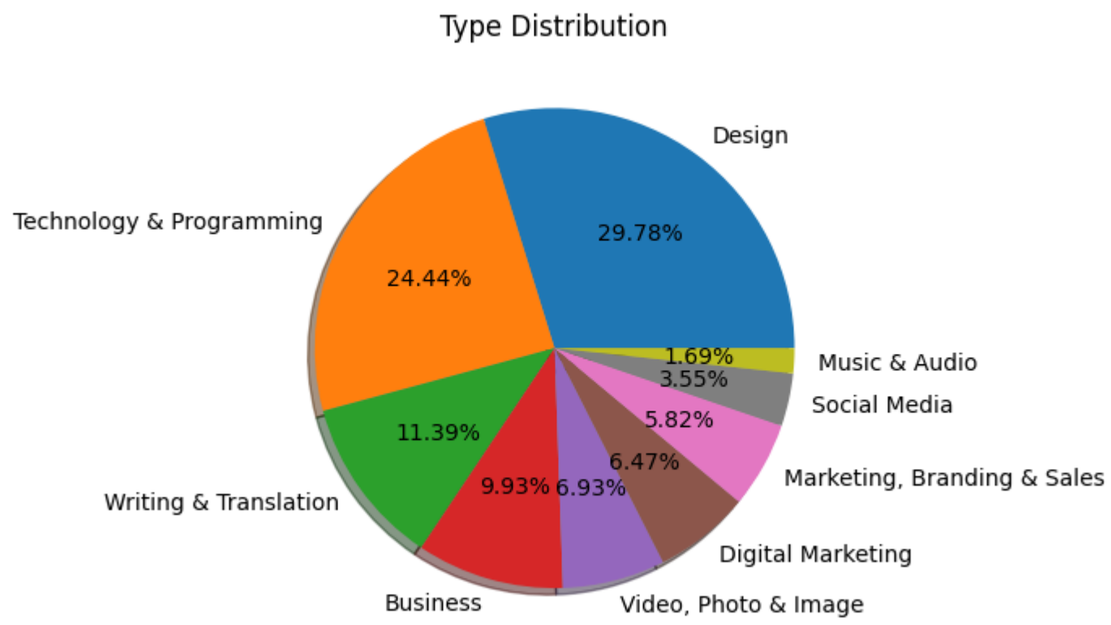
In [36]:
```
plt.figure(figsize=(5,5))

#Plotting the countplot
sns.countplot(x=df['Type'],hue=df['Experience'])

#Adding title,label
plt.title ("Experience wise type distribution")
plt.xlabel('Type')
plt.ylabel('count');
```



From above graph we can understand that the Fixed price has the highest Entry and lower intermediate jobs while hourly type has the highest Expert and lowest intermediate distribution and overall fixed price has most preferred jobs

Understanding the category distibution

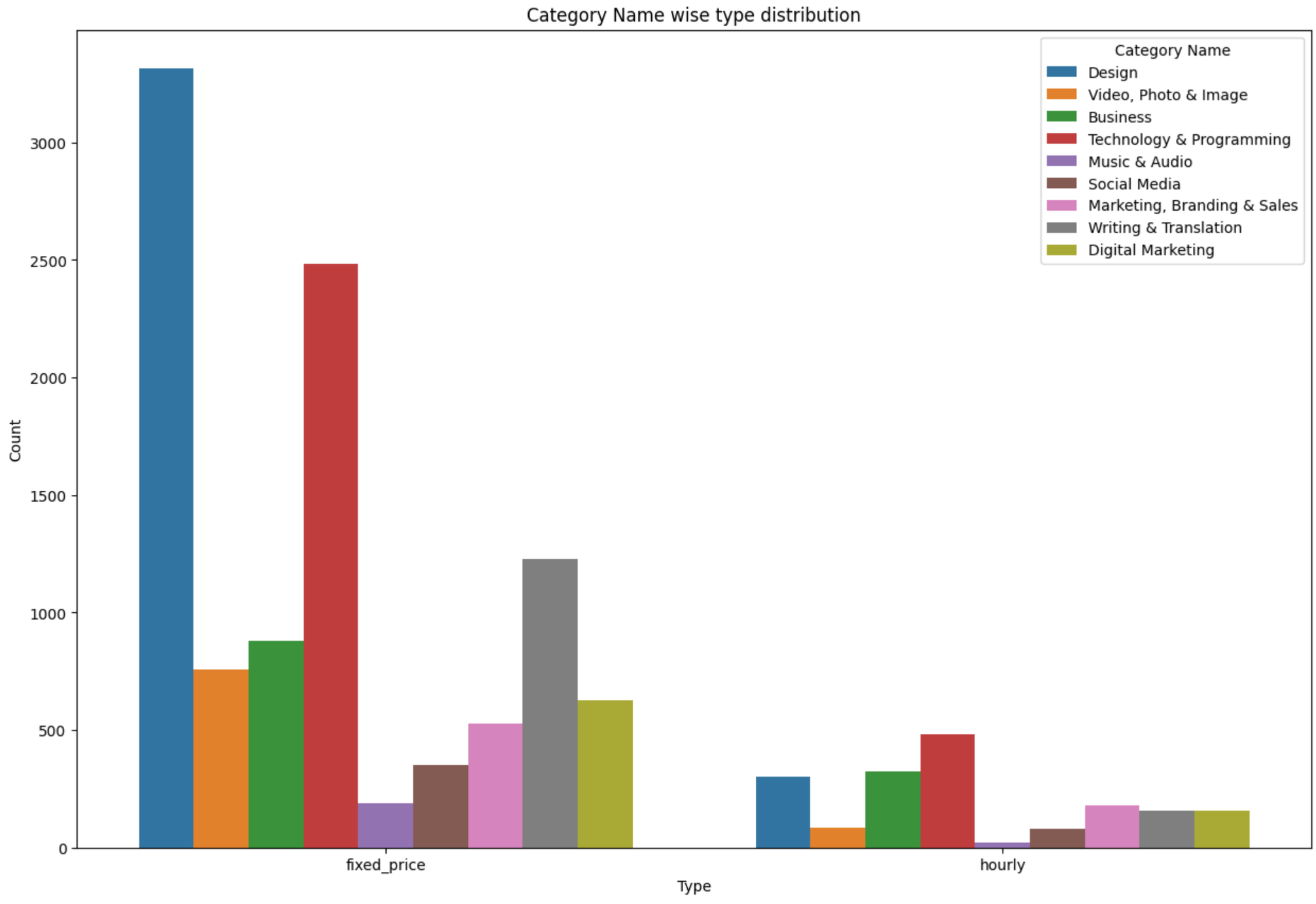
In [37]:
```
category_counts = df['Category Name'].value_counts()

# Plotting the pie chart
plt.pie(category_counts, labels=category_counts.index, autopct='%0.2f%%', shadow=True)

# Adding a title to the chart
plt.title("Type Distribution")

# Show the plot
plt.show()
```



From above graph we can understand that the Design category has the highest and Music & Audio has the lowest distribution

Understanding the Category Name wise type distribution

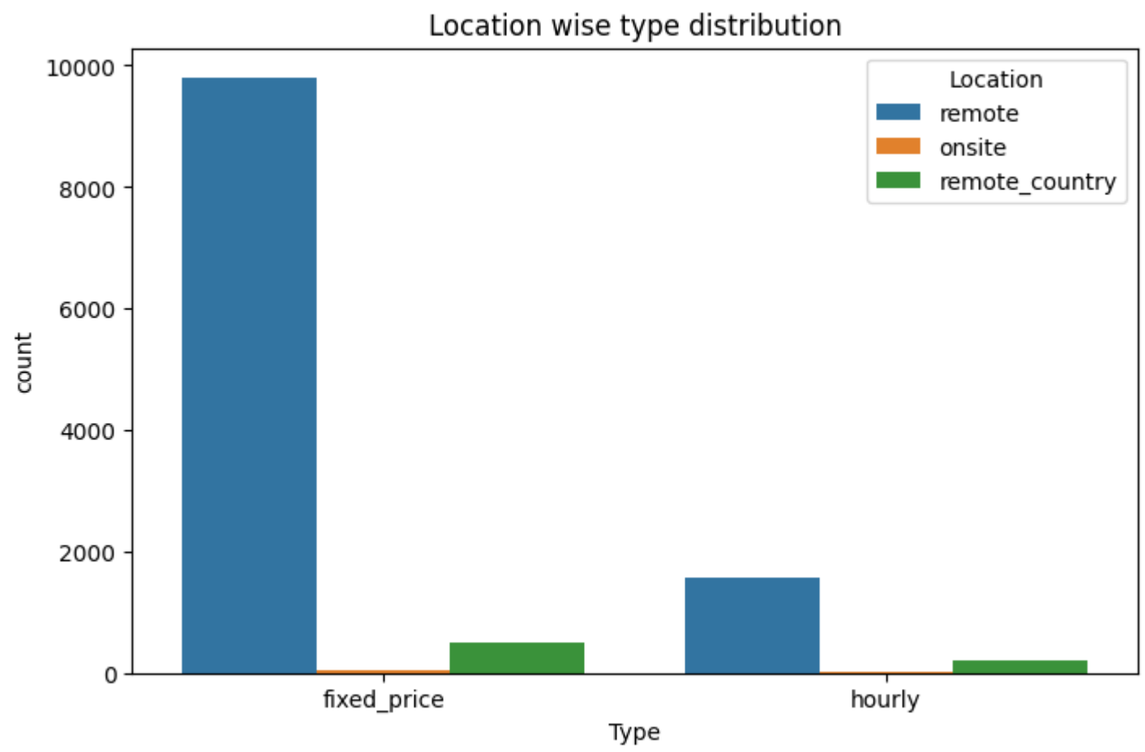
In [38]:
```
plt.figure(figsize=(15,10))

#Plotting the countplot
sns.countplot(x=df['Type'],hue=df['Category Name'])

#Adding title,label
plt.title ("Category Name wise type distribution")
plt.xlabel('Type')
plt.ylabel('Count');
```



From above graph we can understand that the Design category has the highest and Marketing & Branding, Sales has the lowest distribution in fixed price while Technology & programming has the highest and music and audio has the lowest distribution and overall fixed price stands winner in most category jobs

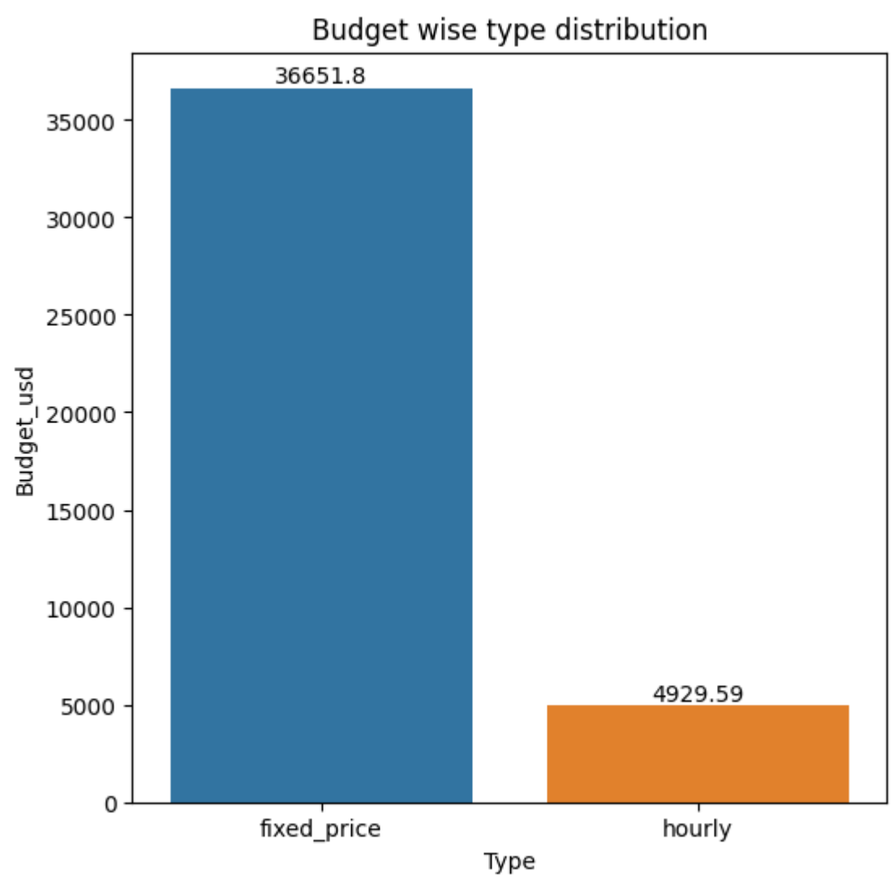
Understanding the Location wise type distribution

In [39]:
```
plt.figure(figsize=(8,5))

#Plotting the countplot
sns.countplot(x=df['Type'],hue=df['Location'])

#Adding title,label
plt.title ("Location wise type distribution")
plt.xlabel('Type')
plt.ylabel('count');
```



Understanding the Budget wise type distribution

In [40]:
```
plt.figure(figsize=(6,6))

grouped = df.groupby(['Type'], as_index=False)['Budget_usd'].sum().sort_values(by='Budget_usd', ascending=False)
# Group the DataFrame 'df' by 'Type' and calculate the sum of 'Budget_usd' for each group, sort the values in descending order, and store the result in 'grouped'.

# Set the title of the plot to "Budget wise type distribution".
plt.title("Budget wise type distribution")

contro = sns.barplot(x='Type', y='Budget_usd', data=grouped)

# Add labels to the bars in the bar plot.
for bars in contro.containers: contro.bar_label(bars)
```

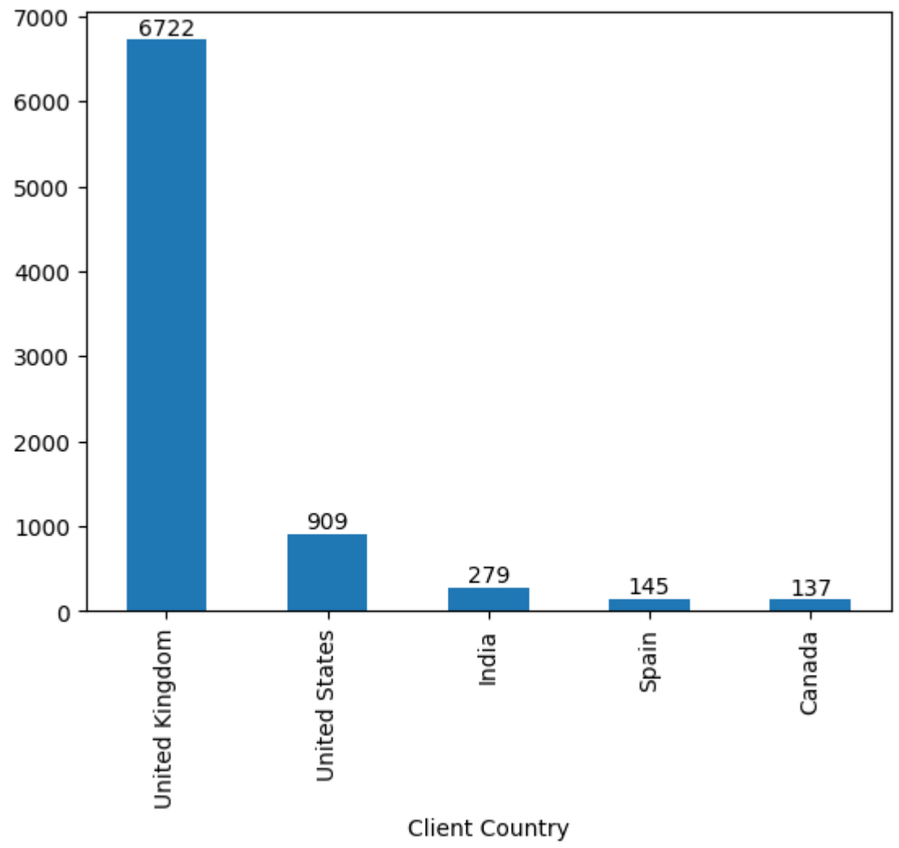


From above graph we can understand that the Budget for fixed price is more than hourly in distribution

In [41]:
```
fixed  = df[df['Type'] == 'fixed_price']
hourly = df[df['Type'] == 'hourly']

top_fixed = fixed['Client Country'].value_counts().head(5)
top_hourly = hourly['Client Country'].value_counts().head(5)
```

Understanding the Top five countries with Fixed hours

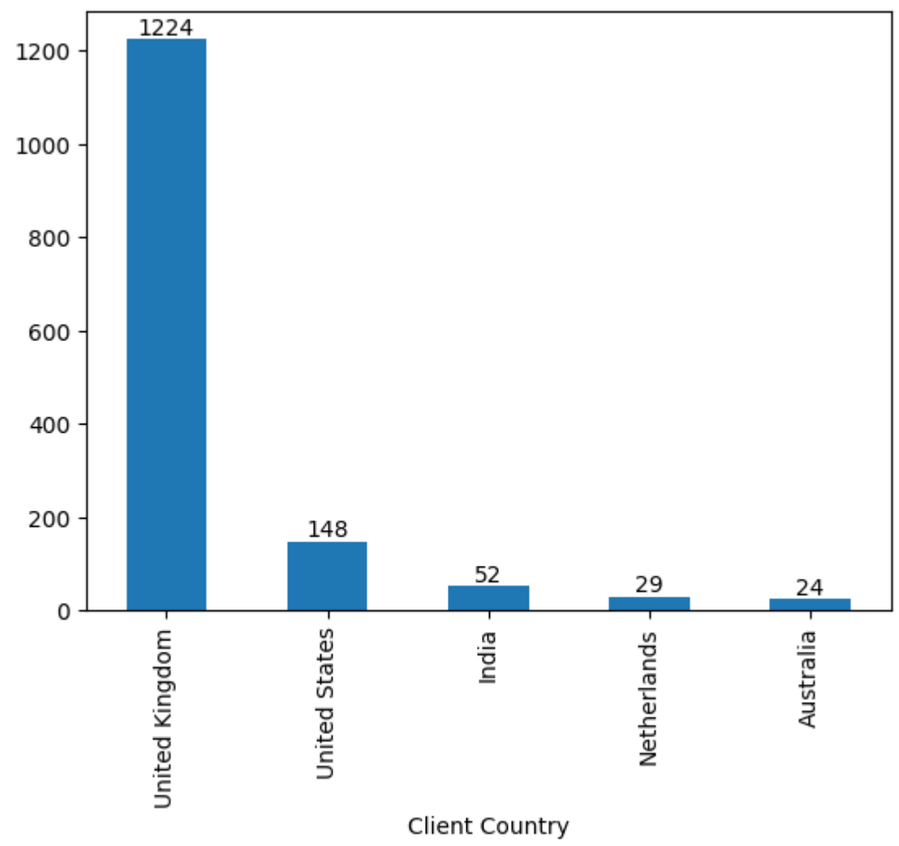
In [42]:
```
countwise=top_fixed.plot(kind='bar')
for bars in countwise.containers:
    countwise.bar_label(bars)
```



From above graph we can understand that the Uk has the highest and Canada has the lowest distribution in fixed hours

Understanding the Top five countries with Hourly hours

In [43]:
```
countwise=top_hourly.plot(kind='bar')
for bars in countwise.containers:
    countwise.bar_label(bars)
```



From above graph we can understand that the Uk has the highest and Australia has the lowest distribution in Hourly hours

In [44]:
```
df.dtypes
```

Out[44]:
```
Title                         object
Category Name                 object
Experience                    object
Sub Category Name             object
Currency                      object
Location                      object
Freelancer Preferred From     object
Type                          object
Description                   object
Client City                   object
Client Country                object
Client Currency               object
Budget_usd                   float64
Client Registration date       int32
Client Registration Month      int32
Client Registration Year       int32
Date Posted in Date            int32
Date Posted in Month           int32
Date Posted in Year            int32
Date Posted in Time           object
dtype: object
```

Categorical data encoding

In [45]:
```
df['Experience'].value_counts()
```

Out[45]:
```
Experience
Expert          5346
Entry           5314
Intermediate    1477
Name: count, dtype: int64
```

In [46]:
```
df['Category Name'].value_counts()
```

Out[46]:
```
Category Name
Design                          3615
Technology & Programming        2966
Writing & Translation           1383
Business                        1205
Video, Photo & Image             841
Digital Marketing                785
Marketing, Branding & Sales      706
Social Media                     431
Music & Audio                    205
Name: count, dtype: int64
```

In [47]:
```
import pandas as pd

# Apply one-hot encoding to 'Experience','Category Name' columns
one_hot_encoded = pd.get_dummies(df[['Category Name','Experience']], dtype=int, prefix=['Category','Experience'])

# Drop the original 'Experience', 'Category Name' columns from the DataFrame
df = df.drop(['Category Name','Experience'], axis=1)

# Concatenate the original DataFrame and the one-hot encoded DataFrame
df = pd.concat([df, one_hot_encoded], axis=1)

# Print the updated DataFrame
print(df.head())
```

```
   Title  Sub Category Name  Currency  Location  Freelancer Preferred From  \
0      1                  0         0         0                          0
1      1                  0         0         0                          0
2      1                  0         0         0                          0
3      1                  0         0         0                          0
5      1                  0         0         0                          0

   Type  Description  Client City  Client Country  Client Currency  \
0     0            0            0               0                0
1     0            0            0               0                0
2     0            0            0               0                0
3     0            0            0               0                0
5     0            0            0               0                0

   Budget_usd  Client Registration date  Client Registration Month  \
0           0                         0                          0
1           0                         0                          0
2           0                         0                          0
3           0                         0                          0
5           0                         0                          0

   Client Registration Year  Date Posted in Date  Date Posted in Month  \
0                         0                    0                     0
1                         0                    0                     0
2                         0                    0                     0
3                         0                    0                     0
5                         0                    0                     0

   Date Posted in Year  Date Posted in Time  Category_Business  \
0                    0                    0                  0
1                    0                    0                  0
2                    0                    0                  0
3                    0                    0                  0
5                    0                    1                  0

   Category_Design  Category_Digital Marketing  \
0                0                           0
1                0                           0
2                1                           0
3                0                           0
5                0                           0

   Category_Marketing, Branding & Sales  Category_Music & Audio  \
0                                     0                       0
1                                     0                       0
2                                     0                       0
3                                     1                       0
5                                     0                       0

   Category_Social Media  Category_Technology & Programming  \
0                      0                                  1
1                      0                                  0
2                      0                                  0
3                      0                                  0
5                      0                                  0

   Category_Video, Photo & Image  Category_Writing & Translation  \
0                              0                              0
1                              1                              0
2                              0                              0
3                              0                              0
5                              0                              0

   Experience_Entry  Experience_Expert  Experience_Intermediate
0                 1                  0                       0
1                 1                  0                       0
2                 1                  0                       0
3                 1                  0                       0
5                 0                  1                       0

[5 rows x 30 columns]
```

In [48]:
```
from sklearn.preprocessing import LabelEncoder

# Identify object columns in the DataFrame
object_columns = df.select_dtypes(include=['object']).columns

# Apply label encoding to each object column
le = LabelEncoder()
for column in object_columns:
    df[column] = le.fit_transform(df[column])
```

In [49]:
```
# Checking correlation using heatmap

plt.figure(figsize=(30,15))
sns.heatmap(df.corr(),cmap='PiYG',annot=True);
```

In [51]:
```
#Dropping below columns as it's not needed for predictions

df.drop(['Date Posted in Year','Currency'],axis=1,inplace=True)
```

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 12137 entries, 0 to 12200
Data columns (total 30 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Title                                 12137 non-null  int32  
 1   Sub Category Name                     12137 non-null  int32  
 2   Currency                              12137 non-null  int32  
 3   Location                              12137 non-null  int32  
 4   Freelancer Preferred From             12137 non-null  int32  
 5   Type                                  12137 non-null  int32  
 6   Description                           12137 non-null  int32  
 7   Client City                           12137 non-null  int32  
 8   Client Country                        12137 non-null  int32  
 9   Client Currency                       12137 non-null  int32  
 10  Budget_usd                            12137 non-null  float64
 11  Client Registration date              12137 non-null  int32  
 12  Client Registration Month             12137 non-null  int32  
 13  Client Registration Year              12137 non-null  int32  
 14  Date Posted in Date                   12137 non-null  int32  
 15  Date Posted in Month                  12137 non-null  int32  
 16  Date Posted in Year                   12137 non-null  int32  
 17  Date Posted in Time                   12137 non-null  int32  
 18  Category_Business                     12137 non-null  int32  
 19  Category_Design                       12137 non-null  int32  
 20  Category_Digital Marketing            12137 non-null  int32  
 21  Category_Marketing, Branding & Sales  12137 non-null  int32  
 22  Category_Music & Audio                12137 non-null  int32  
 23  Category_Social Media                 12137 non-null  int32  
 24  Category_Technology & Programming     12137 non-null  int32  
 25  Category_Video, Photo & Image         12137 non-null  int32  
 26  Category_Writing & Translation        12137 non-null  int32  
 27  Experience_Entry                      12137 non-null  int32  
 28  Experience_Expert                     12137 non-null  int32  
 29  Experience_Intermediate               12137 non-null  int32  
dtypes: float64(1), int32(29)
memory usage: 1.8 MB
```

Feature Selection

We do not require additional features as they do not significantly contribute to the results. Moreover, including extra features can increase the computational complexity of our algorithms. Therefore, we will only select the important features for our analysis.

In [52]:
```
from sklearn.feature_selection import SelectKBest,chi2
x = df.drop(columns=['Type'], axis=1)
y = df['Type']
```

By using SelectKBest with the chi-squared test, we can identify the most relevant features for classification. The chi-squared test measures the independence between each feature and the target variable. It helps in selecting features that have the strongest relationship with the target variable.

By performing feature selection and separating the features from the target variable, we are preparing the data for classification modeling. This allows us to focus on the most informative features and ensure that the model receives the appropriate input for training and prediction.

In [53]:
```
x.shape
```

Out[53]: (12137, 27)

In [54]:
```
y.shape
```

Out[54]: (12137,)

In [55]:
```
#Createing a SelectKBest object with the chi-squared test as the scoring function.

skb = SelectKBest(chi2)

# To apply the feature selection process to the feature matrix x and the target variable y.
# The fit method calculates the chi-squared scores for each feature and determines the best features based on the scores.

best_scores = skb.fit(x,y)
```

In [56]:
```
best_scores.scores_ #retrieves the chi-squared scores calculated by the SelectKBest object.
```

Out[56]:
```
array([1.28463563e+04, 1.76731053e+02, 4.63814646e+00, 1.21029647e+03,
       3.51551841e+04, 4.20236640e+00, 7.17484729e+01, 2.30194915e+00,
       2.69589034e+02, 5.34057543e+00, 6.17864686e+00, 1.62110750e-03,
       3.51034491e-01, 1.61856845e-01, 7.58672176e+00, 1.42529277e+02,
       1.17328076e+02, 1.83855086e+01, 6.37999799e+01, 4.83415388e+00,
       5.73748092e+00, 5.88369911e+00, 1.41868740e+01, 1.35024567e+01,
       1.60809922e+01, 1.45439533e+01, 1.23125679e-01])
```

In [57]:
```
skb_score = pd.DataFrame(best_scores.scores_) #creates a DataFrame (skb_score) to store the chi-squared scores.
skb_columns = pd.DataFrame(x.columns) # creates a DataFrame (skb_columns) to store the feature names (column names of x).

list_of_scores = pd.concat([ skb_columns, skb_score], axis = 1)## Getting the above two columns together through concat function.

list_of_scores.columns = ['Features', 'Scores'] #assigns column names ('Features' and 'Scores') to the DataFrame list_of_scores

list_of_scores
```

Out[57]:

	Features	Scores
0	Title	12846.356283
1	Sub Category Name	176.731053
2	Location	4.638146
3	Freelancer Preferred From	1210.296471
4	Description	35155.184113
5	Client City	4.202366
6	Client Country	71.748473
7	Client Currency	2.301949
8	Budget_usd	269.589034
9	Client Registration date	5.340575
10	Client Registration Month	6.178647
11	Client Registration Year	0.001621
12	Date Posted in Date	0.351034
13	Date Posted in Month	0.161857
14	Date Posted in Time	7.586722
15	Category_Business	142.529277
16	Category_Design	117.328076
17	Category_Digital Marketing	18.385509
18	Category_Marketing, Branding & Sales	63.799980
19	Category_Music & Audio	4.834154
20	Category_Social Media	5.737481
21	Category_Technology & Programming	5.883699
22	Category_Video, Photo & Image	14.186874
23	Category_Writing & Translation	13.502457
24	Experience_Entry	16.080992
25	Experience_Expert	14.543953
26	Experience_Intermediate	0.123126

The purpose of these steps is to calculate the chi-squared scores for each feature and create a DataFrame (list_of_scores) that provides a summary of the feature importance based on these scores. By examining the chi-squared scores, you can identify the features that have the strongest relationship with the target variable, indicating their relevance for classification.

In [58]:
```
list_of_scores.nlargest(26, 'Scores') #sorting the values
```

Out[58]:

	Features	Scores
4	Description	35155.184113
0	Title	12846.356283
3	Freelancer Preferred From	1210.296471
8	Budget_usd	269.589034
1	Sub Category Name	176.731053
15	Category_Business	142.529277
16	Category_Design	117.328076
6	Client Country	71.748473
18	Category_Marketing, Branding & Sales	63.799980
17	Category_Digital Marketing	18.385509
24	Experience_Entry	16.080992
25	Experience_Expert	14.543953
22	Category_Video, Photo & Image	14.186874
23	Category_Writing & Translation	13.502457
14	Date Posted in Time	7.586722
10	Client Registration Month	6.178647
21	Category_Technology & Programming	5.883699
20	Category_Social Media	5.737481
9	Client Registration date	5.340575
19	Category_Music & Audio	4.834154
2	Location	4.638146
5	Client City	4.202366
7	Client Currency	2.301949
12	Date Posted in Date	0.351034
13	Date Posted in Month	0.161857
26	Experience_Intermediate	0.123126

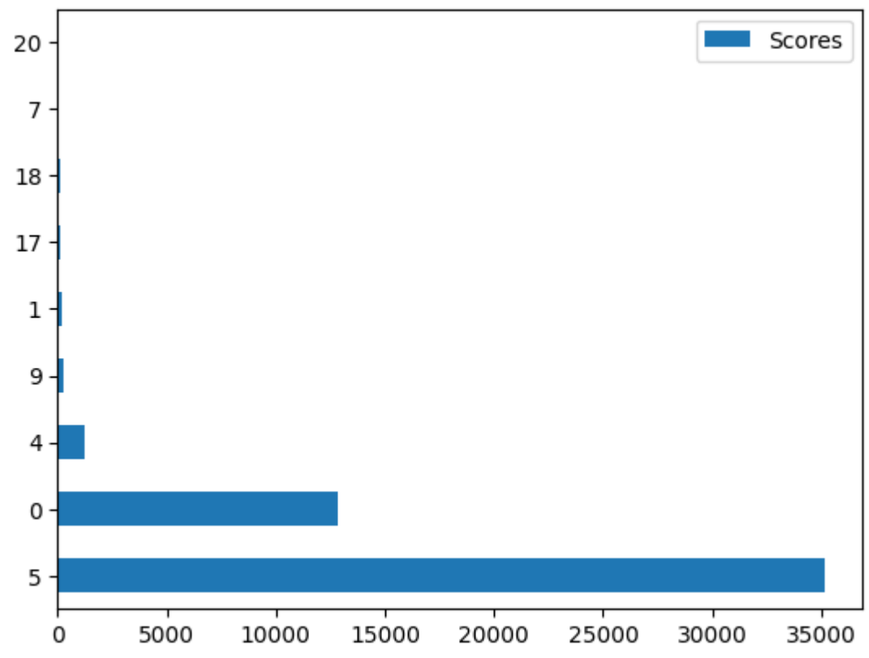
In [59]:
```
list_of_scores.nlargest(10, 'Scores') #sorting top 9 scores
```

Out[59]:

	Features	Scores
4	Description	35155.184113
0	Title	12846.356283
3	Freelancer Preferred From	1210.296471
8	Budget_usd	269.589034
1	Sub Category Name	176.731053
15	Category_Business	142.529277
16	Category_Design	117.328076
6	Client Country	71.748473
18	Category_Marketing, Branding & Sales	63.799980
17	Category_Digital Marketing	18.385509

In [58]:
```
list_of_scores.nlargest(9, 'Scores')
list_of_scores.nlargest(9, 'Scores').plot(kind = 'barh')
```

Out[58]: <Axes: >



In [60]:
```
y = y = df['Type']
```

In [61]:
```
y.shape
```

Out[61]: (12137,)

This univariate feature selection did not give us clear understanding of important feature. Now we will use ExtraTreesClassifier from the sklearn library for Feature Importance Selection. This class implements a meta estimator that fits a number of randomized decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

In [62]:
```
from sklearn.ensemble import ExtraTreesClassifier
ft_imp = ExtraTreesClassifier()
ft_imp.fit(x, y)
```

Out[62]: ExtraTreesClassifier()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

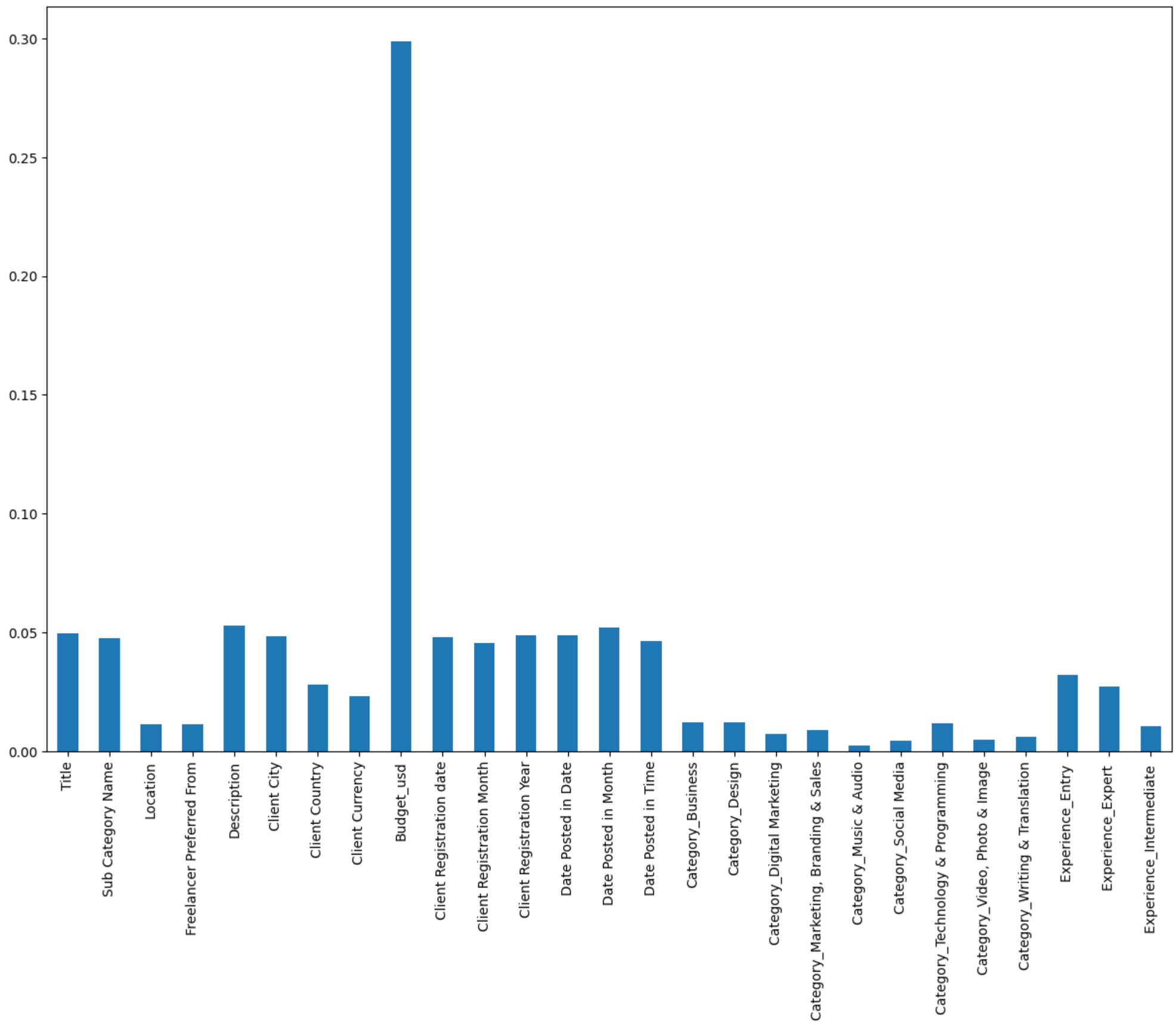
This classifier is an ensemble learning method that fits multiple randomized decision trees on different sub-samples of the dataset and combines their predictions to determine feature importance.

The purpose of these steps in classification is to determine the importance of features for predicting the target variable. The ExtraTreesClassifier algorithm constructs an ensemble of decision trees and evaluates the impact of each feature on the accuracy of the predictions. After fitting the ExtraTreesClassifier model, you can access the feature importances through the feature_importances_ attribute of the trained model object (ft_imp in this case). These importances can be used to identify the most influential features in the classification task, helping to focus on the most informative aspects of the data for prediction

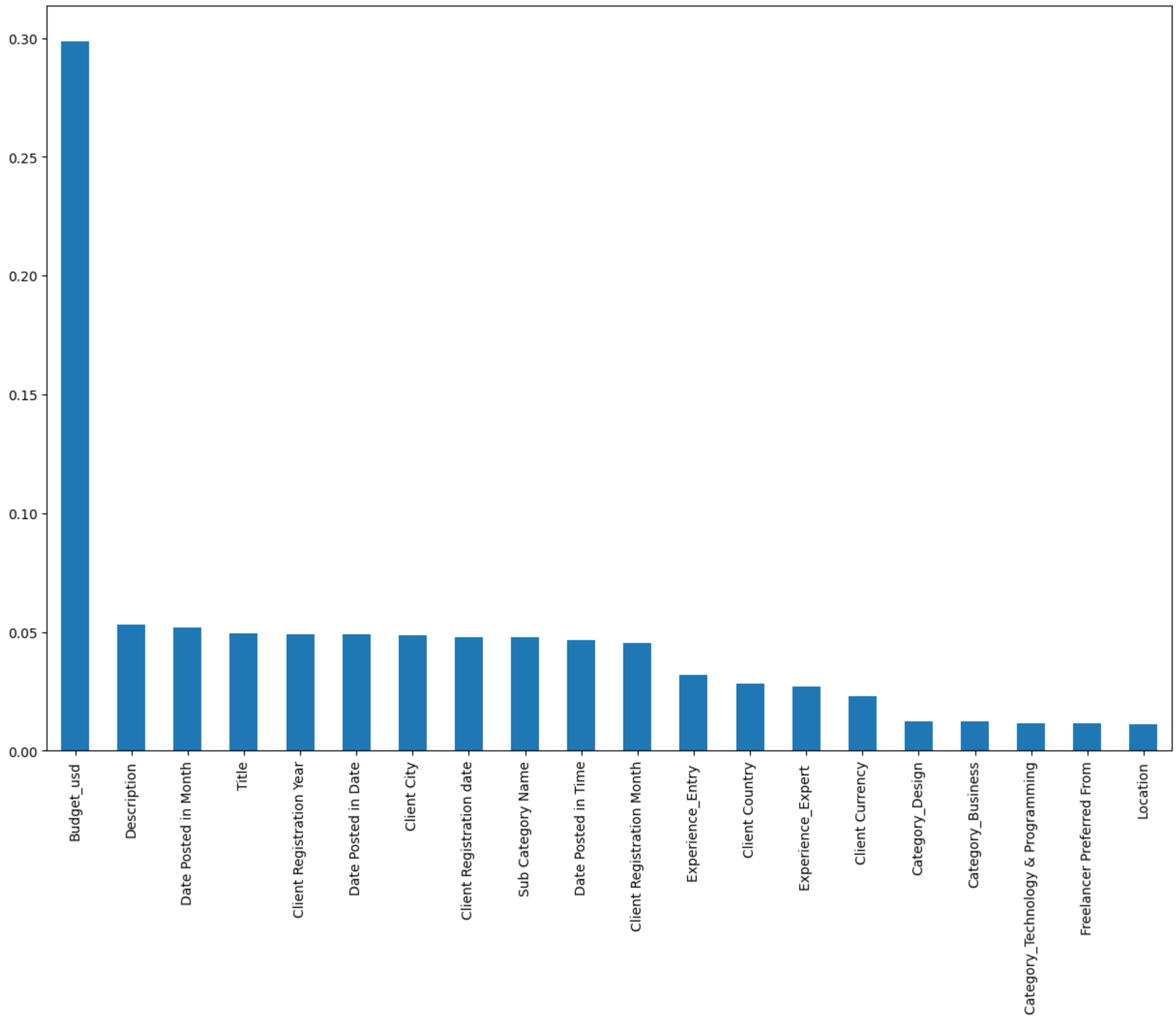
In [63]:
```
ft_imp.feature_importances_
```

Out[63]:
```
array([0.04949509, 0.04767762, 0.01113669, 0.01147775, 0.05303798,
       0.04842918, 0.02819446, 0.0229442 , 0.29858897, 0.04792267,
       0.04542166, 0.04884293, 0.04883446, 0.0519696 , 0.04651507,
       0.01220292, 0.01227332, 0.00707933, 0.00882832, 0.00233911,
       0.00419253, 0.01162097, 0.00496472, 0.0061652 , 0.03207252,
       0.02721576, 0.01055696])
```

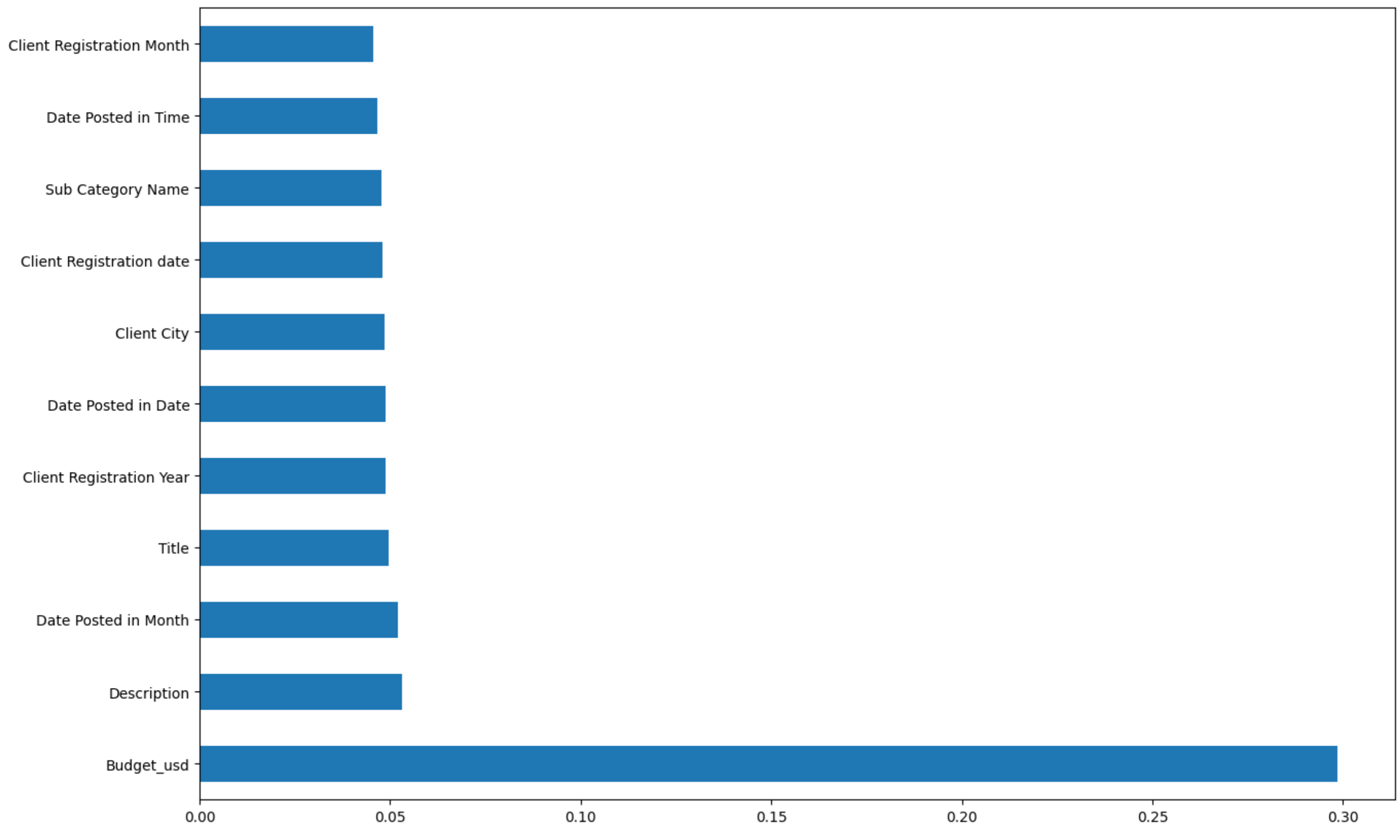
In [67]:
```
plt.figure(figsize=(15,10))
feature_importance = pd.Series(ft_imp.feature_importances_, index=x.columns)
feature_importance.plot(kind='bar')
plt.show()
```



In [68]:
```
plt.figure(figsize=(15,10))
feature_importance = pd.Series(ft_imp.feature_importances_, index=x.columns)
feature_importance.nlargest(20).plot(kind='bar')
plt.show()
```



In [69]:
```
plt.figure(figsize=(15,10))
feature_importance = pd.Series(ft_imp.feature_importances_, index=x.columns)
feature_importance.nlargest(11).plot(kind='barh')
plt.show()
```



After Multiple checks we found that the top 12 will be the only features useful to us, we do not need the rest 9 columns and hence will be dropping them from our dataset before applying any model.

In [93]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 12120 entries, 0 to 12200
Data columns (total 19 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Title                      12120 non-null  float64
 1   Sub Category Name          12120 non-null  float64
 2   Type                       12120 non-null  float64
 3   Description                12120 non-null  float64
 4   Client City                12120 non-null  float64
 5   Client Country             12120 non-null  float64
 6   Client Currency            12120 non-null  float64
 7   Budget_usd                 12120 non-null  float64
 8   Client Registration date   12120 non-null  float64
 9   Client Registration Month  12120 non-null  float64
 10  Client Registration Year   12120 non-null  float64
 11  Date Posted in Date        12120 non-null  float64
 12  Date Posted in Month       12120 non-null  float64
 13  Date Posted in Time        12120 non-null  float64
 14  Category_Music & Audio     12120 non-null  float64
 15  Category_Social Media      12120 non-null  float64
 16  Experience_Entry           12120 non-null  float64
 17  Experience_Expert          12120 non-null  float64
 18  Experience_Intermediate    12120 non-null  float64
dtypes: float64(19)
memory usage: 1.8 MB
```

In [71]:
```
columns_to_drop = ['Location',
                   'Category_Design',
                   'Freelancer Preferred From',
                   'Category_Technology & Programming',
                   'Category_Business',
                   'Category_Marketing, Branding & Sales',
                   'Category_Digital Marketing',
                   'Category_Writing & Translation',
                   'Category_Video, Photo & Image',
                    ]

df.drop(columns=columns_to_drop, inplace=True)
```

In [72]:
```
df.columns
```

Out[72]:
```
Index(['Title', 'Sub Category Name', 'Type', 'Description', 'Client City',
       'Client Country', 'Client Currency', 'Budget_usd',
       'Client Registration date', 'Client Registration Month',
       'Client Registration Year', 'Date Posted in Date',
       'Date Posted in Month', 'Date Posted in Time', 'Category_Music & Audio',
       'Category_Social Media', 'Experience_Entry ', 'Experience_Expert ',
       'Experience_Intermediate '],
      dtype='object')
```

Normalising the data

In [73]:
```
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Specify the column(s) to be scaled
column_to_scale = df.columns

# Apply Min-Max scaling to the selected column(s)
df[column_to_scale] = scaler.fit_transform(df[column_to_scale])

print(df)
```

```
12197                    0.0                    0.0                1.0
12198                    0.0                    0.0                0.0
12199                    0.0                    0.0                1.0
12200                    0.0                    0.0                0.0

       Experience_Expert   Experience_Intermediate
0                    0.0                      0.0
1                    0.0                      0.0
2                    0.0                      0.0
3                    0.0                      0.0
5                    1.0                      0.0
...                  ...                      ...
12196                0.0                      0.0
12197                0.0                      0.0
12198                0.0                      1.0
12199                0.0                      0.0
12200                1.0                      0.0

[12137 rows x 19 columns]
```

Checking Duplicated Values before training the model

In [75]:
```
df.duplicated().sum()
```

Out[75]:
```
17
```

In [76]:
```
df.drop_duplicates(inplace=True)
```

In [77]:
```
df.duplicated().sum()
```

Out[77]:
```
0
```

In [78]:
```
print('Shape of our dataframe is:',df.shape)
```

```
Shape of our dataframe is: (12120, 19)
```

Split the data into x and y

In [79]:
```
x = df.drop(columns=['Type'], axis=1)
y = df['Type']
```

In [80]:
```
y.value_counts()
```

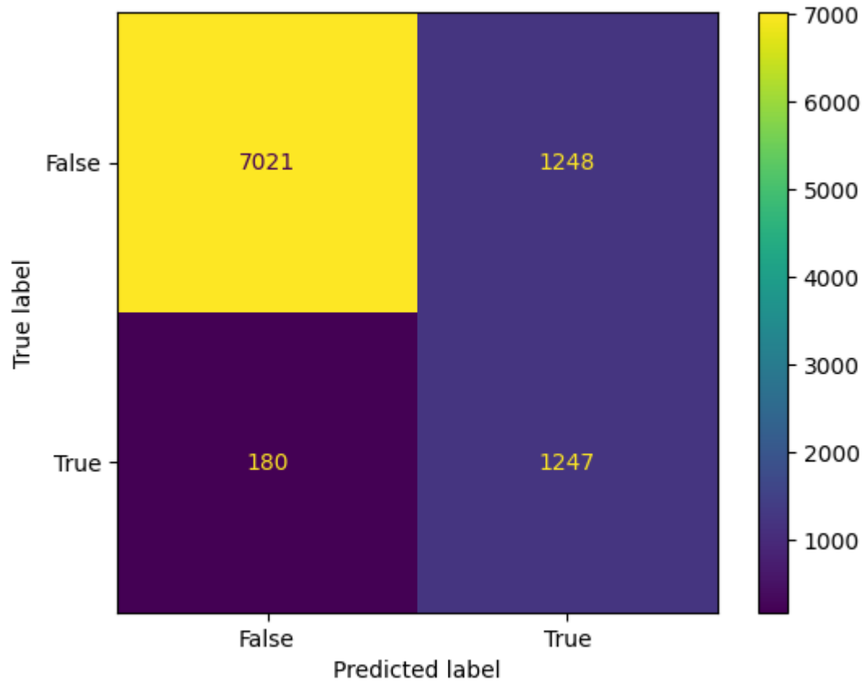
Out[80]:
```
Type
0.0    10336
1.0     1784
Name: count, dtype: int64
```

In [81]:
```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x,y,train_size=0.8,stratify=y)
```

Creating a Classification model

Logistic Regression

In [82]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import matplotlib.pyplot as plt

#Train the model
model = LogisticRegression(class_weight='balanced')
model.fit(xtrain, ytrain)

#Making predictions on training and testing
trainpred = model.predict(xtrain)
testpred = model.predict(xtest)

cm = metrics.confusion_matrix(ytrain, trainpred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm,display_labels=[False, True])

cm_display.plot()
plt.show()
```



True Positive (TP): The model predicted 1247 instances as true (positive class), and they were indeed true.

True Negative (TN): The model predicted 7021 instances as false (negative class), and they were indeed false.

False Positive (FP): The model incorrectly predicted 1248 instances as true, but they were actually false.

False Negative (FN): The model incorrectly predicted 180 instances as false, but they were actually true.

In [83]:
```
print("Training Data:")
print(metrics.classification_report(ytrain, trainpred))
print("Testing Data:")
print(metrics.classification_report(ytest, testpred))
```

```
Training Data:
              precision    recall  f1-score   support

         0.0       0.98      0.85      0.91      8269
         1.0       0.50      0.87      0.64      1427

    accuracy                           0.85      9696
   macro avg       0.74      0.86      0.77      9696
weighted avg       0.91      0.85      0.87      9696

Testing Data:
              precision    recall  f1-score   support

         0.0       0.97      0.86      0.91      2067
         1.0       0.51      0.87      0.64       357

    accuracy                           0.86      2424
   macro avg       0.74      0.86      0.78      2424
weighted avg       0.91      0.86      0.87      2424
```

In [84]:
```
model.score(xtest,ytest)
```

Out[84]: 0.8589108910891089

Decision Tree

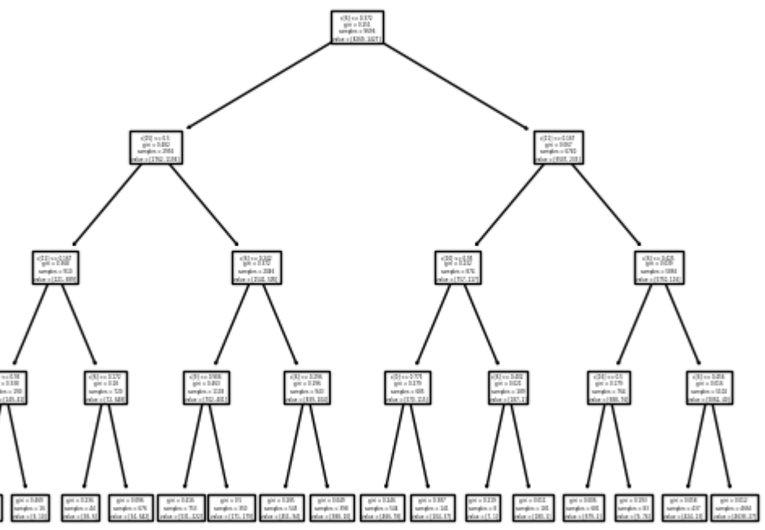
In [85]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

#Train the model
model = DecisionTreeClassifier(max_depth=4)
model.fit(xtrain, ytrain)

#Making predictions on training and testing
trainpred = model.predict(xtrain)
testpred = model.predict(xtest)

print("Training Data:")
print(classification_report(ytrain, trainpred))
print("Testing Data:")
print(classification_report(ytest, testpred))
```

```
Training Data:
              precision    recall  f1-score   support

         0.0       0.94      0.97      0.96      8269
         1.0       0.80      0.63      0.71      1427

    accuracy                           0.92      9696
   macro avg       0.87      0.80      0.83      9696
weighted avg       0.92      0.92      0.92      9696

Testing Data:
              precision    recall  f1-score   support

         0.0       0.94      0.98      0.96      2067
         1.0       0.82      0.65      0.73       357

    accuracy                           0.93      2424
   macro avg       0.88      0.81      0.84      2424
weighted avg       0.92      0.93      0.92      2424
```

In [86]:
```
model.score(xtest,ytest)
```

Out[86]:
```
0.9278052805280528
```

In [87]:
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plot_tree(model)
plt.show()
```



Random Forest

In [88]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report

#Train the model
rf = RandomForestClassifier(random_state=40)
rf.fit(xtrain, ytrain)

#Evaluating the model
trainpred = rf.predict(xtrain)
testpred = rf.predict(xtest)

print("Training Data:")
print(classification_report(ytrain, trainpred))
print("\nTesting Data:")
print(classification_report(ytest, testpred))
```

```
Training Data:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      8269
         1.0       1.00      1.00      1.00      1427

    accuracy                           1.00      9696
   macro avg       1.00      1.00      1.00      9696
weighted avg       1.00      1.00      1.00      9696


Testing Data:
              precision    recall  f1-score   support

         0.0       0.94      0.99      0.96      2067
         1.0       0.89      0.65      0.75       357

    accuracy                           0.94      2424
   macro avg       0.92      0.82      0.86      2424
weighted avg       0.93      0.94      0.93      2424
```

In [86]:
```
model.score(xtest,ytest)
```

Out[86]:
```
0.9187293729372937
```

Support Vector Machine

In [89]:
```
from sklearn.svm import SVC
from sklearn.metrics import classification_report

#Train the model
model = SVC(kernel='poly')
model.fit(xtrain, ytrain)

#Making predictions on training and testing
trainpred=model.predict(xtrain)
testpred = model.predict(xtest)

print("Training Data:")
print(classification_report(ytrain, trainpred))
print("Testing Data:")
print(classification_report(ytest, testpred))
```

```
Training Data:
              precision    recall  f1-score   support

         0.0       0.92      0.99      0.96      8269
         1.0       0.92      0.50      0.65      1427

    accuracy                           0.92      9696
   macro avg       0.92      0.75      0.80      9696
weighted avg       0.92      0.92      0.91      9696

Testing Data:
              precision    recall  f1-score   support

         0.0       0.92      0.99      0.96      2067
         1.0       0.92      0.51      0.66       357

    accuracy                           0.92      2424
   macro avg       0.92      0.75      0.81      2424
weighted avg       0.92      0.92      0.91      2424
```

In [88]:
```
model.score(xtest,ytest)
```

Out[88]:
```
0.9195544554455446
```

XG Boost

In [90]:
```
import xgboost as xgb
from sklearn.metrics import classification_report

#Train the model
model = xgb.XGBClassifier()
model.fit(xtrain, ytrain)

#Making predictions on training and testing
trainpred=model.predict(xtrain)
testpred = model.predict(xtest)

print("Training Data:")
print(classification_report(ytrain, trainpred))
print("Testing Data:")
print(classification_report(ytest, testpred))
```

```
Training Data:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      8269
         1.0       1.00      1.00      1.00      1427

    accuracy                           1.00      9696
   macro avg       1.00      1.00      1.00      9696
weighted avg       1.00      1.00      1.00      9696

Testing Data:
              precision    recall  f1-score   support

         0.0       0.95      0.97      0.96      2067
         1.0       0.82      0.72      0.77       357

    accuracy                           0.94      2424
   macro avg       0.89      0.85      0.86      2424
weighted avg       0.93      0.94      0.93      2424
```

In [91]:
```
model.score(x,y)
```

Out[91]:
```
0.9868811881188119
```

Result:-

Based on the above results, the XGBOOST model performs the best among the models evaluated. It achieves the highest accuracy for the testing data, indicating superior overall performance in accurately predicting the target variable compared to the other models. The XGBOOST model's high accuracy suggests that it correctly predicts the target variable in a larger proportion of instances in the testing data, making it the most reliable and accurate model for this task. In conclusion, based on the results, the XGBOOST model is the best-performing model for the task at hand, surpassing the performance of the Random Forest, Decision Tree, SVM, and Logistic Regression models in terms of accuracy for the testing data.