1. Perform necessary EDA on the data

In [1]:
```
# we import the dataset with the help of pandas library
import pandas as pd

df = pd.read_csv('Freelance_Projects.csv')
df.head()
```

Out[1]:

	Title	Category Name	Experience	Sub Category Name	Currency	Budget	Location	Freelancer Preferred From	
0	I need an HTML/CSS design for an email template	Technology & Programming	Entry ($)	Email Template Development	GBP	40	remote	ALL	fixed_
1	Virtual Assistance with Wordpress experience	Design	Expert ($$$)	Web Design	USD	110	remote	ALL	fixed_
2	Hot Tub Hire Company Website design	Design	Intermediate ($$)	Web Design	GBP	88	remote	ALL	fixed_
3	Farm House Design	Design	Intermediate ($$)	3D Design	EUR	154	remote	ALL	fixed_
4	Adapt Greek Contract	Writing & Translation	Expert ($$$)	Business Writing	EUR	138	remote	ALL	fixed_

In [2]:
```
# This will provide the shape of the dataset

df.shape
```

Out[2]:
```
(9747, 17)
```

In [3]:
```
#This gives the details about the dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9747 entries, 0 to 9746
Data columns (total 17 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Title                      9747 non-null   object
 1   Category Name              9747 non-null   object
 2   Experience                 9747 non-null   object
 3   Sub Category Name          9747 non-null   object
 4   Currency                   9747 non-null   object
 5   Budget                     9747 non-null   int64
 6   Location                   9747 non-null   object
 7   Freelancer Preferred From  9747 non-null   object
 8   Type                       9747 non-null   object
 9   Date Posted                9747 non-null   object
 10  Description                9747 non-null   object
 11  Duration                   1286 non-null   object
 12  Client Registration Date   9747 non-null   object
 13  Client City                9747 non-null   object
 14  Client Country             9747 non-null   object
 15  Client Currency            9747 non-null   object
 16  Client Job Title           3708 non-null   object
dtypes: int64(1), object(16)
memory usage: 1.3+ MB
```

In [4]:
```
#This will provide the column names

df.columns.values
```

Out[4]:
```
array(['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'], dtype=object)
```

In [5]:
```
#Here we get the columns which has null values

missing_values = df.isnull().sum()
```

In [6]:
```
print(missing_values)
```

```
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                    8461
Client Registration Date       0
Client City                    0
Client Country                 0
Client Currency                0
Client Job Title            6039
dtype: int64
```

In [7]:
```
#droped the columns which has Missing data and stored it in a new dataframe

newdf = df.dropna(axis=1)
print(df.shape)
print(newdf.shape)
```

```
(9747, 17)
(9747, 15)
```

In [8]:
```
#prints the new dataframe column names
newdf.columns
```

Out[8]:
```
Index(['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'],
      dtype='object')
```

In [9]:
```
# This is used to create a list of column names in a pandas DataFrame df that have the

obj_col = newdf.columns[newdf.dtypes=='object']
```

In [10]:
```
#LabelEncoder is used for transforming categorical data into numerical data
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

for i in obj_col:
    le = LabelEncoder()
    newdf[i]=le.fit_transform(newdf[i])
```

```
C:\Users\Prabha\AppData\Local\Temp\ipykernel_2396\1974061257.py:8: SettingWithCopyWar
ning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  newdf[i]=le.fit_transform(newdf[i])
```

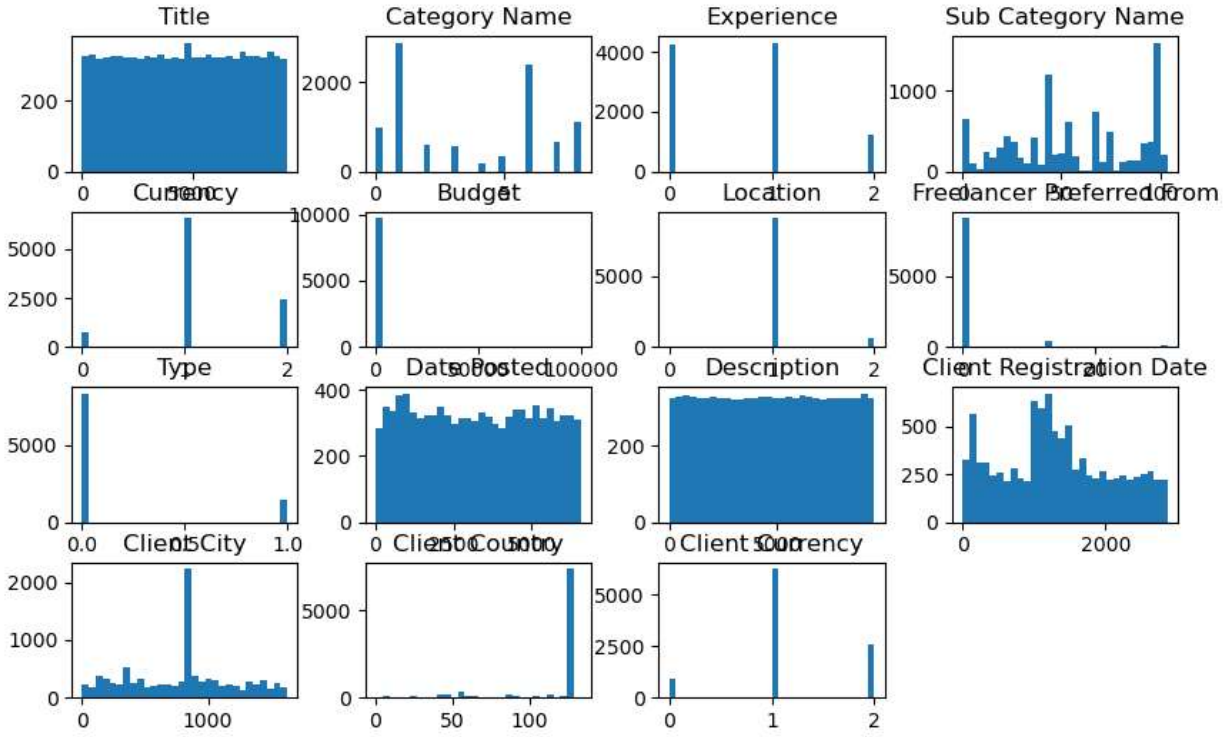
In [11]:
```
#Here the new dataframe has all numeeric values
newdf.head()
```

Out[11]:

	Title	Category Name	Experience	Sub Category Name	Currency	Budget	Location	Freelancer Preferred From	Type	Date Posted	Des
0	3819	6	0	30	1	40	1	1	0	6581	
1	8071	1	1	99	2	110	1	1	0	6581	
2	3237	1	2	99	1	88	1	1	0	6581	
3	2587	1	2	0	0	154	1	1	0	6581	
4	397	8	1	13	0	138	1	1	0	6581	

In [12]:
```
newdf.hist(grid=False, bins=30, figsize=(10,6))
```

Out[12]:
```
array([[<AxesSubplot:title={'center':'Title'}>,
        <AxesSubplot:title={'center':'Category Name'}>,
        <AxesSubplot:title={'center':'Experience'}>,
        <AxesSubplot:title={'center':'Sub Category Name'}>],
       [<AxesSubplot:title={'center':'Currency'}>,
        <AxesSubplot:title={'center':'Budget'}>,
        <AxesSubplot:title={'center':'Location'}>,
        <AxesSubplot:title={'center':'Freelancer Preferred From'}>],
       [<AxesSubplot:title={'center':'Type'}>,
        <AxesSubplot:title={'center':'Date Posted'}>,
        <AxesSubplot:title={'center':'Description'}>,
        <AxesSubplot:title={'center':'Client Registration Date'}>],
       [<AxesSubplot:title={'center':'Client City'}>,
        <AxesSubplot:title={'center':'Client Country'}>,
        <AxesSubplot:title={'center':'Client Currency'}>, <AxesSubplot:>]],
      dtype=object)
```



In [13]:
```
newdf.agg(['skew', 'kurtosis']).transpose()
```

Out[13]:

	skew	kurtosis
Title	-0.006983	-1.197581
Category Name	0.130030	-1.597613
Experience	0.478192	-0.805186
Sub Category Name	-0.050926	-1.196706
Currency	0.100064	-0.008636
Budget	45.253803	2373.613474
Location	2.770459	11.093274
Freelancer Preferred From	4.798010	25.499113
Type	1.985777	1.943710
Date Posted	0.023321	-1.234968
Description	-0.000316	-1.202652
Client Registration Date	0.177888	-0.865987
Client City	0.011855	-0.765145
Client Country	-1.948380	2.487631
Client Currency	-0.017111	-0.228782

In [14]:
```
#describe() method to get a quick summary of the datasets
#including the count, mean, standard deviation, minimum, and maximum values for each c
newdf.describe()
```

Out[14]:

	Title	Category Name	Experience	Sub Category Name	Currency	Budget	Location	I
count	9747.000000	9747.000000	9747.00000	9747.000000	9747.000000	9747.000000	9747.000000	97
mean	4635.987586	3.721145	0.68852	56.246743	1.177696	221.601108	1.053452	
std	2669.731509	2.821600	0.67923	31.996623	0.541357	1769.402837	0.250005	
min	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	
25%	2326.500000	1.000000	0.00000	30.000000	1.000000	30.000000	1.000000	
50%	4646.000000	3.000000	1.00000	52.000000	1.000000	80.000000	1.000000	
75%	6953.500000	6.000000	1.00000	91.000000	2.000000	150.000000	1.000000	
max	9256.000000	8.000000	2.00000	104.000000	2.000000	99999.000000	2.000000	

2. Use machine learning to create clusters of similar projects.

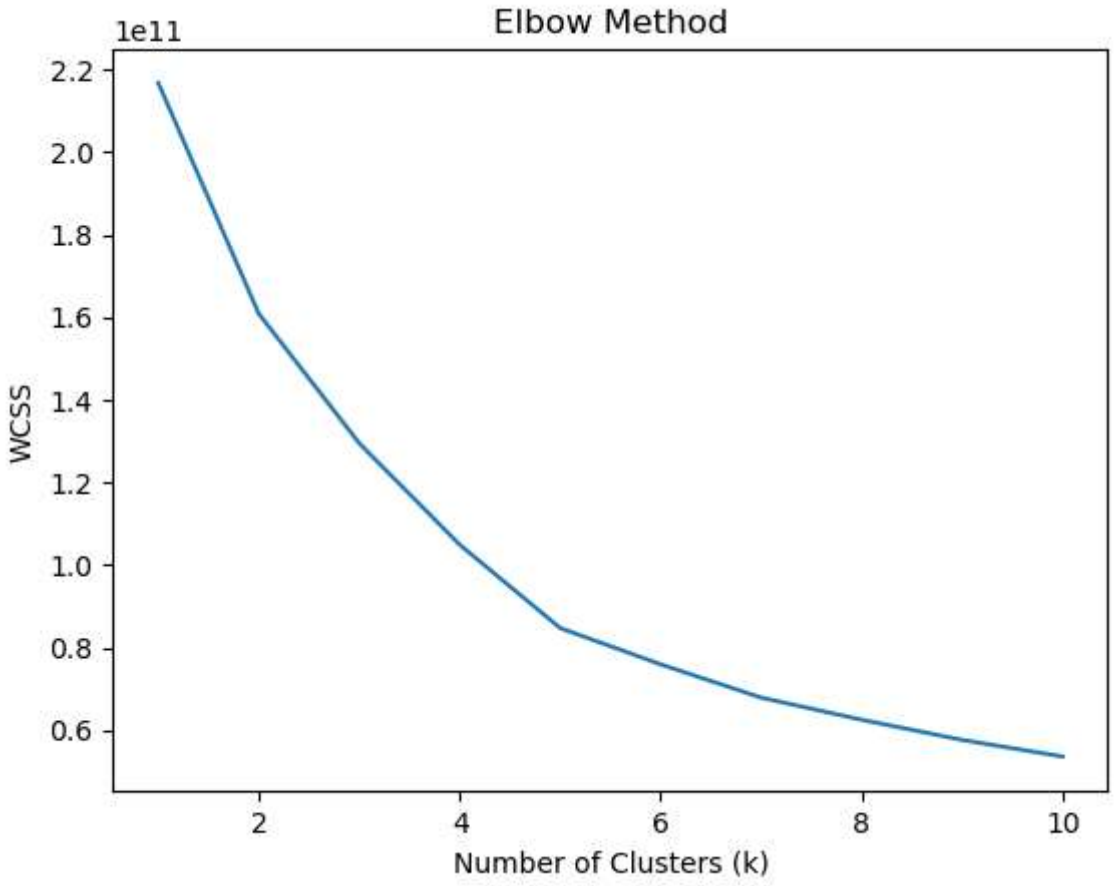
In [15]:
```
#Indexcing all rows and all columns and printed 1st 5 rows
x=newdf.iloc[:, :].values
x[:5]
```

Out[15]:
```
array([[3819,    6,    0,   30,    1,   40,    1,    1,    0, 6581, 2148,
        1593,  855,  127,    1],
       [8071,    1,    1,   99,    2,  110,    1,    1,    0, 6581, 1963,
        2493,  652,  127,    1],
       [3237,    1,    2,   99,    1,   88,    1,    1,    0, 6581, 2348,
         567,  310,  127,    1],
       [2587,    1,    2,    0,    0,  154,    1,    1,    0, 6581, 3808,
        2118,  714,  106,    2],
       [ 397,    8,    1,   13,    0,  138,    1,    1,    0, 6581, 9295,
        1073, 1448,   50,    0]], dtype=int64)
```

In [16]:
```
#Performing KMeans on the dataset
from sklearn.cluster import KMeans
wcss = {}

for i in range(1,11):
    model=KMeans(n_clusters=i)
    model.fit(x)
    wcss[i]=model.inertia_

wcss
```

Out[16]:
```
{1: 216731141335.02878,
 2: 160847379675.9344,
 3: 129464842983.31108,
 4: 104883213503.1194,
 5: 84703789233.165,
 6: 75911978354.59525,
 7: 67840443025.44423,
 8: 62489966150.44221,
 9: 57597555825.46936,
 10: 53578729968.583176}
```

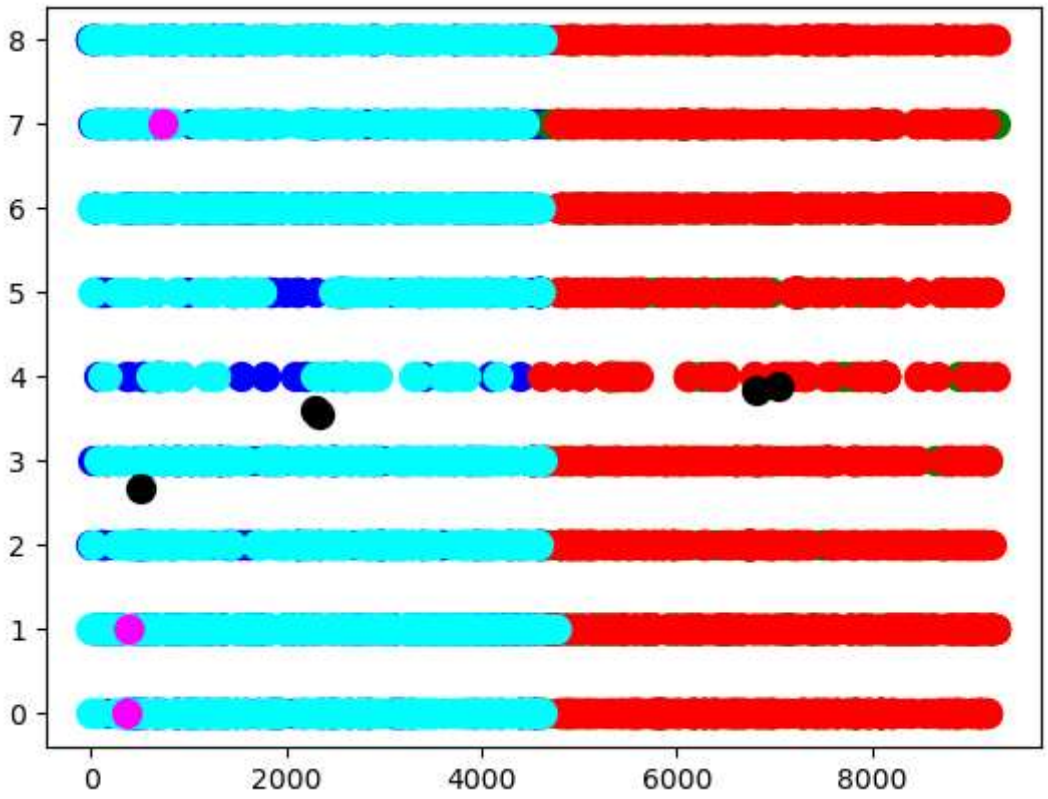
In [17]:
```
#Created Elbow method
import matplotlib.pyplot as plt

plt.plot(wcss.keys(), wcss.values())
plt.title('Elbow Method')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.show()
```



In [18]:
```
#Using KMeans performed clustering on similar datapoints
model = KMeans(n_clusters=5)
pred=model.fit_predict(x)
pred[:5]
```

Out[18]:
```
array([3, 1, 3, 3, 0])
```

In [19]:
```
plt.scatter(x[pred==0, 0], x[pred==0, 1], s=100, c='blue', label='Cluster 1')
plt.scatter(x[pred==1, 0], x[pred==1, 1], s=100, c='green', label='Cluster 2')
plt.scatter(x[pred==2, 0], x[pred==2, 1], s=100, c='red', label='Cluster 3')
plt.scatter(x[pred==3, 0], x[pred==3, 1], s=100, c='cyan', label='Cluster 4')
plt.scatter(x[pred==4, 0], x[pred==4, 1], s=100, c='magenta', label='Cluster 5')
plt.scatter(model.cluster_centers_[:,0],model.cluster_centers_[:,1], s = 100, c='blacl
plt.show()
```



3. Create a regression model to predict the budget and evaluate it.

In [20]:
```
#used the new dataframe to create the regression model
newdf.head()
```

Out[20]:

	Title	Category Name	Experience	Sub Category Name	Currency	Budget	Location	Freelancer Preferred From	Type	Date Posted	Des
0	3819	6	0	30	1	40	1	1	0	6581	
1	8071	1	1	99	2	110	1	1	0	6581	
2	3237	1	2	99	1	88	1	1	0	6581	
3	2587	1	2	0	0	154	1	1	0	6581	
4	397	8	1	13	0	138	1	1	0	6581	

In [21]:
```
newdf.columns
```

Out[21]:
```
Index(['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'],
      dtype='object')
```

In [22]:
```
#drop 3 columns 'Date Posted, Description, Client Registration Date' because of Multic
#stored in a new variable

newdf1 = newdf.drop(columns=newdf.columns[[9,10,11]], axis=1)
```

In [23]:
```
newdf1.columns
```

Out[23]:
```
Index(['Title', 'Category Name', 'Experience', 'Sub Category Name', 'Currency',
       'Budget', 'Location', 'Freelancer Preferred From', 'Type',
       'Client City', 'Client Country', 'Client Currency'],
      dtype='object')
```

In [24]:
```
#indexed all row and all columns to predict against budget column

x1 = newdf1.iloc[:, :]
y1 = newdf1['Budget']
```

In [25]:
```
#Splited the data into training and testing

from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x1,
                            y1, test_size=0.9, random_state=1)
```

In [26]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(xtrain, ytrain)
pred=model.predict(xtest)
pred[:5]
```

Out[26]:
```
array([  50., 1000.,  100.,   12.,   90.])
```

In [27]:
```
model.intercept_
```

Out[27]:
```
1.4210854715202004e-13
```

In [28]:
```
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(ytest, pred)
mse
```

Out[28]:
```
6.383745518491144e-25
```

In [29]:
```
r2 = r2_score(ytest, pred)
r2
```

Out[29]:
```
1.0
```

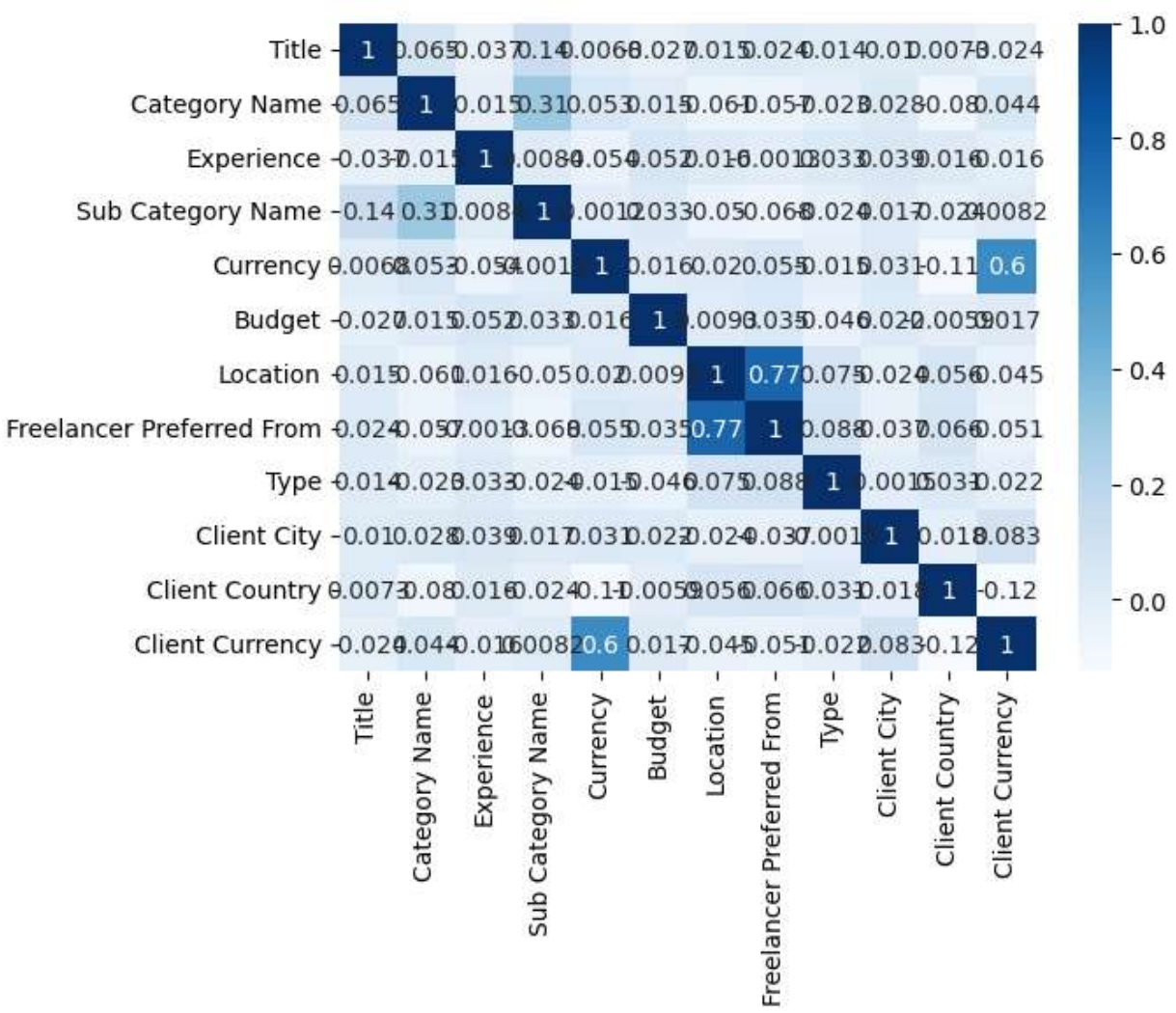
In [30]:
```
newdf1.corr()
```

Out[30]:

	Title	Category Name	Experience	Sub Category Name	Currency	Budget	Location	Freelancer Preferred From	
Title	1.000000	0.064943	-0.036914	0.137881	0.006836	-0.027032	0.015209	0.023668	(
Category Name	0.064943	1.000000	-0.015184	0.305009	0.053468	0.014720	-0.061486	-0.056683	-(
Experience	-0.036914	-0.015184	1.000000	0.008404	-0.053720	0.051635	0.016484	-0.001303	(
Sub Category Name	0.137881	0.305009	0.008404	1.000000	-0.001234	0.032781	-0.049929	-0.067628	-(
Currency	0.006836	0.053468	-0.053720	-0.001234	1.000000	0.016084	0.020030	0.055245	-(
Budget	-0.027032	0.014720	0.051635	0.032781	0.016084	1.000000	0.009314	0.034573	-(
Location	0.015209	-0.061486	0.016484	-0.049929	0.020030	0.009314	1.000000	0.765261	(
Freelancer Preferred From	0.023668	-0.056683	-0.001303	-0.067628	0.055245	0.034573	0.765261	1.000000	(
Type	0.013540	-0.023001	0.033005	-0.024294	-0.015426	-0.045817	0.075210	0.087948	
Client City	0.010129	0.028070	0.039391	0.016973	0.030732	0.021542	-0.023724	-0.037284	-(
Client Country	0.007320	-0.080429	0.016306	-0.023550	-0.110963	-0.005901	0.056282	0.066402	(
Client Currency	-0.023626	0.043925	-0.016312	0.008152	0.603049	0.017034	-0.045067	-0.050898	-(

In [31]:
```
import seaborn as sns
sns.heatmap(newdf1.corr(), annot=True, cmap='Blues')
```

Out[31]:
```
<AxesSubplot:>
```



4. Create a classification model to predict the value of the Type column and evaluate it.

In [32]:
```
#used the new dataframe to create the regression model

newdf.head()
```

Out[32]:

	Title	Category Name	Experience	Sub Category Name	Currency	Budget	Location	Freelancer Preferred From	Type	Date Posted	Des
0	3819	6	0	30	1	40	1	1	0	6581	
1	8071	1	1	99	2	110	1	1	0	6581	
2	3237	1	2	99	1	88	1	1	0	6581	
3	2587	1	2	0	0	154	1	1	0	6581	
4	397	8	1	13	0	138	1	1	0	6581	

In [33]:
```
##Splited the data into training and testing

from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x1,
                            y1, test_size=0.7, random_state=50)
```

In [34]:
```
#installed lazypredict to check which model is best suits to our data

!pip install lazypredict
```

```
Requirement already satisfied: lazypredict in c:\users\prabha\anaconda3\lib\site-pack
ages (0.2.12)
Requirement already satisfied: click in c:\users\prabha\anaconda3\lib\site-packages
(from lazypredict) (8.0.4)
Requirement already satisfied: pandas in c:\users\prabha\anaconda3\lib\site-packages
(from lazypredict) (1.4.4)
Requirement already satisfied: tqdm in c:\users\prabha\anaconda3\lib\site-packages (f
rom lazypredict) (4.64.1)
Requirement already satisfied: xgboost in c:\users\prabha\anaconda3\lib\site-packages
(from lazypredict) (1.7.5)
Requirement already satisfied: joblib in c:\users\prabha\anaconda3\lib\site-packages
(from lazypredict) (1.1.0)
Requirement already satisfied: lightgbm in c:\users\prabha\anaconda3\lib\site-package
s (from lazypredict) (3.3.5)
Requirement already satisfied: scikit-learn in c:\users\prabha\anaconda3\lib\site-pac
kages (from lazypredict) (1.0.2)
Requirement already satisfied: colorama in c:\users\prabha\anaconda3\lib\site-package
s (from click->lazypredict) (0.4.5)
Requirement already satisfied: wheel in c:\users\prabha\anaconda3\lib\site-packages
(from lightgbm->lazypredict) (0.37.1)
Requirement already satisfied: numpy in c:\users\prabha\anaconda3\lib\site-packages
(from lightgbm->lazypredict) (1.21.5)
Requirement already satisfied: scipy in c:\users\prabha\anaconda3\lib\site-packages
(from lightgbm->lazypredict) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\prabha\anaconda3\lib
\site-packages (from scikit-learn->lazypredict) (2.2.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\prabha\anaconda3\lib\site-pac
kages (from pandas->lazypredict) (2022.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\prabha\anaconda3\li
b\site-packages (from pandas->lazypredict) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\users\prabha\anaconda3\lib\site-package
s (from python-dateutil>=2.8.1->pandas->lazypredict) (1.16.0)
```

In [35]:
```
#By the table we can see the the time taken for DecisionTreeClassifier is less than th

from lazypredict.Supervised import LazyClassifier

clf = LazyClassifier(verbose=0,
                     ignore_warnings=True)
models, predictions = clf.fit(xtrain, xtest,
                              ytrain, ytest)
models
```

```
100%|██████████████████████████████████████████████████████████████████████████████
██| 29/29 [01:10<00:00,  2.44s/it]
```

Out[35]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LinearDiscriminantAnalysis	0.96	0.56	None	0.95	0.08
DecisionTreeClassifier	0.93	0.41	None	0.92	0.06
BaggingClassifier	0.92	0.38	None	0.91	0.43
GaussianNB	0.88	0.29	None	0.86	0.11
RandomForestClassifier	0.68	0.14	None	0.64	2.23
ExtraTreesClassifier	0.48	0.11	None	0.45	2.55
ExtraTreeClassifier	0.29	0.08	None	0.29	0.04
LabelSpreading	0.13	0.04	None	0.13	2.60
LabelPropagation	0.13	0.04	None	0.13	0.96
NearestCentroid	0.01	0.03	None	0.01	0.05
AdaBoostClassifier	0.28	0.02	None	0.15	3.42
KNeighborsClassifier	0.12	0.02	None	0.10	0.40
LinearSVC	0.16	0.01	None	0.10	7.66
LogisticRegression	0.14	0.01	None	0.09	3.39
BernoulliNB	0.14	0.01	None	0.08	0.05
SVC	0.14	0.01	None	0.09	9.60
PassiveAggressiveClassifier	0.06	0.01	None	0.04	0.39
CalibratedClassifierCV	0.14	0.01	None	0.08	30.34
SGDClassifier	0.08	0.01	None	0.06	0.74
Perceptron	0.06	0.01	None	0.05	0.41
RidgeClassifier	0.12	0.01	None	0.06	0.20
RidgeClassifierCV	0.12	0.01	None	0.06	0.08
LGBMClassifier	0.06	0.00	None	0.03	4.29
DummyClassifier	0.06	0.00	None	0.01	0.03

In [36]:
```
#Hence used DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
model.fit(xtrain, ytrain)
pred = model.predict(xtest)
pred[:5]
```

Out[36]:
```
array([ 80, 106, 600, 160,  20], dtype=int64)
```

In [37]:
```
from sklearn.metrics import accuracy_score

accuracy_score(ytest, pred)
```

Out[37]:
```
0.9259856368166496
```

In [38]:
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

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