Q1 The data under the filename “IPL” provides the sold price of players in auction based on different variables. The data description is given in filename “IPL-Description”. Write the Python Programming to

a. Perform a exploratory Data Analysis on the input features

b. Build a Multiple Regression Model to predict the “Sold Price” based on all the independent variables. Comment on the model based on the R-Square value and p-value for the parameters. [Note: train size = 0.8, random state = 42] c. Print the Heat Map for numerical variables

In [32]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
```

In [33]:
```
df = pd.read_csv("/Users/shubham/Desktop/Assigment/IPL.csv")
```

In [34]:
```
df.head(10)
```

Out[34]:

	Sl.NO.	PLAYER NAME	AGE	PLAYING ROLE	T-RUNS	T-WKTS	ODI-RUNS-S	ODI-SR-B	ODI-WKTS	ODI-SR-BL	...	HIGH SCORE	AVE	SR -B	SIXERS	RUNS-C	WKTS	AVE-BL	ECON	SR-BL	SOLD PRICE
0	1	Abdulla, YA	2	Allrounder	0	0	0	0.00	0	0.0	...	0	0.00	0.00	0	307	15	20.47	8.90	13.93	50000
1	2	Abdur Razzak	2	Bowler	214	18	657	71.41	185	37.6	...	0	0.00	0.00	0	29	0	0.00	14.50	0.00	50000
2	3	Agarkar, AB	2	Bowler	571	58	1269	80.62	288	32.9	...	39	18.56	121.01	5	1059	29	36.52	8.81	24.90	350000
3	4	Ashwin, R	1	Bowler	284	31	241	84.56	51	36.8	...	11	5.80	76.32	0	1125	49	22.96	6.23	22.14	850000
4	5	Badrinath, S	2	Batsman	63	0	79	45.93	0	0.0	...	71	32.93	120.71	28	0	0	0.00	0.00	0.00	800000
5	6	Bailey, GJ	2	Batsman	0	0	172	72.26	0	0.0	...	48	21.00	95.45	0	0	0	0.00	0.00	0.00	50000
6	7	Balaji, L	2	Bowler	51	27	120	78.94	34	42.5	...	15	4.33	72.22	1	1342	52	25.81	7.98	19.40	500000
7	8	Bollinger, DE	2	Bowler	54	50	50	92.59	62	31.3	...	16	21.00	165.88	1	693	37	18.73	7.22	15.57	700000
8	9	Botha, J	2	Allrounder	83	17	609	85.77	72	53.0	...	67	30.45	114.73	3	610	19	32.11	6.85	28.11	950000
9	10	Boucher, MV	2	W. Keeper	5515	1	4686	84.76	0	0.0	...	50	28.14	127.51	13	0	0	0.00	0.00	0.00	450000

10 rows × 22 columns

EDA

In [ ]:

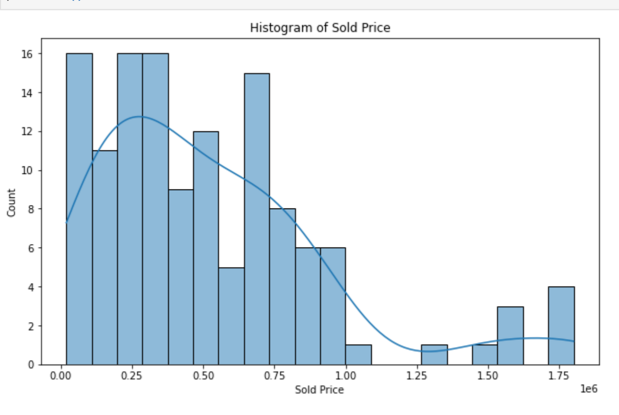
In [37]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 130 entries, 0 to 129
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Sl.NO.         130 non-null    int64  
 1   PLAYER NAME    130 non-null    object 
 2   AGE            130 non-null    int64  
 3   PLAYING ROLE   130 non-null    object 
 4   T-RUNS         130 non-null    int64  
 5   T-WKTS         130 non-null    int64  
 6   ODI-RUNS-S     130 non-null    int64  
 7   ODI-SR-B       130 non-null    float64
 8   ODI-WKTS       130 non-null    int64  
 9   ODI-SR-BL      130 non-null    float64
 10  CAPTAINCY EXP  130 non-null    int64  
 11  RUNS-S         130 non-null    int64  
 12  HIGH SCORE     130 non-null    int64  
 13  AVE            130 non-null    float64
 14  SR -B          130 non-null    float64
 15  SIXERS         130 non-null    int64  
 16  RUNS-C         130 non-null    int64  
 17  WKTS           130 non-null    int64  
 18  AVE-BL         130 non-null    float64
 19  ECON           130 non-null    float64
 20  SR-BL          130 non-null    float64
 21  SOLD PRICE     130 non-null    int64  
dtypes: float64(7), int64(13), object(2)
memory usage: 22.5+ KB
None
```

In [38]:
```
print(df.describe())
```

```
           Sl.NO.         AGE        T-RUNS      T-WKTS    ODI-RUNS-S  \
count  130.000000  130.000000    130.000000  130.000000    130.000000   
mean    65.500000    2.092308   2166.715385   66.530769   2508.738462   
std     37.671829    0.576627   3305.646757  142.676855   3582.205625   
min      1.000000    1.000000      0.000000    0.000000      0.000000   
25%     33.250000    2.000000     25.500000    0.000000     73.250000   
50%     65.500000    2.000000    542.500000    7.000000    835.000000   
75%     97.750000    2.000000   3002.250000   47.500000   3523.500000   
max    130.000000    3.000000  15470.000000  800.000000  18426.000000   

         ODI-SR-B    ODI-WKTS   ODI-SR-BL  CAPTAINCY EXP      RUNS-S  \
count  130.000000  130.000000  130.000000     130.000000  130.000000   
mean    71.164385   76.076923   34.033846       0.315385  514.246154   
std     25.898440  111.205070   26.751749       0.466466  615.226335   
min      0.000000    0.000000    0.000000       0.000000    0.000000   
25%     65.650000    0.000000    0.000000       0.000000   39.000000   
50%     78.225000   18.500000   36.600000       0.000000  172.000000   
75%     86.790000  106.000000   45.325000       1.000000  925.250000   
max    116.660000  534.000000  150.000000       1.000000 2254.000000   

       HIGH SCORE         AVE       SR -B      SIXERS       RUNS-C  \
count  130.000000  130.000000  130.000000  130.000000   130.000000   
mean    47.430769   18.719308  111.053462   17.692308   475.523077   
std     36.403624   11.094224   35.928907   23.828146   558.314049   
min      0.000000    0.000000    0.000000    0.000000     0.000000   
25%     16.000000    9.825000   98.237500    1.000000     0.000000   
50%     35.500000   18.635000  118.510000    6.000000   297.000000   
75%     73.750000   27.872500  129.102500   29.750000   689.250000   
max    158.000000   50.110000  235.490000  129.000000  1975.000000   

             WKTS      AVE-BL        ECON       SR-BL    SOLD PRICE  
count  130.000000  130.000000  130.000000  130.000000  1.300000e+02  
mean    17.169231   23.110231    6.204462   17.382615  5.212231e+05  
std     21.816763   20.802057    4.941531   15.273422  4.068074e+05  
min      0.000000    0.000000    0.000000    0.000000  2.000000e+04  
25%      0.000000    0.000000    0.000000    0.000000  2.250000e+05  
50%      8.500000   24.785000    7.380000   19.935000  4.375000e+05  
75%     23.750000   35.580000    8.247500   26.212500  7.000000e+05  
max     83.000000  126.300000   38.110000  100.200000  1.800000e+06  
```

In [39]:
```
plt.figure(figsize=(10, 6))
sns.histplot(df['SOLD PRICE'], bins=20, kde=True)
plt.xlabel('Sold Price')
plt.title('Histogram of Sold Price')
plt.show()
```



In [43]:
```
df['PLAYING ROLE']=df['PLAYING ROLE'].astype('category')
df['PLAYING ROLE'] = df['PLAYING ROLE'].cat.codes
```

In [44]:
```
import statsmodels.api as sm
Y = df["SOLD PRICE"]
X = df.drop(["PLAYER NAME","SOLD PRICE"], axis = 1)
```

In [45]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random_state=42)
len(X_test)
```

Out[45]:
```
26
```

In [46]:
```
df_lm=sm.OLS(Y_train , X_train).fit()
X_train=sm.add_constant(X_train)
```

In [47]:
```
print(df_lm.params)
df_lm.summary2()
```

```
Sl.NO.              2000.585345
AGE               -23402.963801
PLAYING ROLE        9656.413024
T-RUNS               -21.783374
T-WKTS              -455.518178
ODI-RUNS-S            26.688992
ODI-SR-B             357.433887
ODI-WKTS             808.654078
ODI-SR-BL           -562.311463
CAPTAINCY EXP     229322.160851
RUNS-S               142.569953
HIGH SCORE         -7143.138041
AVE                26776.029195
SR -B              -1295.208349
SIXERS              5036.224828
RUNS-C                28.714956
WKTS                4615.686703
AVE-BL              4683.189205
ECON               -5279.191657
SR-BL              -2714.991584
dtype: float64
```

Out[47]:

Model:	OLS	Adj. R-squared (uncentered):	0.737
Dependent Variable:	SOLD PRICE	AIC:	2963.1796
Date:	2023-10-08 21:24	BIC:	3016.0674
No. Observations:	104	Log-Likelihood:	-1461.6
Df Model:	20	F-statistic:	15.60
Df Residuals:	84	Prob (F-statistic):	1.98e-20
R-squared (uncentered):	0.788	Scale:	1.1674e+11

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
Sl.NO.	2000.5853	941.7142	2.1244	0.0366	127.8834	3873.2873
AGE	-23402.9638	53892.1403	-0.4343	0.6652	-130573.3950	83767.4674
PLAYING ROLE	9656.4130	38003.0269	0.2541	0.8000	-65916.7737	85229.5998
T-RUNS	-21.7834	30.6859	-0.7099	0.4797	-82.8056	39.2388
T-WKTS	-455.5182	565.4818	-0.8055	0.4228	-1580.0408	669.0044
ODI-RUNS-S	26.6890	30.8346	0.8656	0.3892	-34.6290	88.0069
ODI-SR-B	357.4339	1523.4862	0.2346	0.8151	-2672.1856	3387.0534
ODI-WKTS	808.6541	708.2768	1.1417	0.2568	-599.8319	2217.1401
ODI-SR-BL	-562.3115	1491.3955	-0.3770	0.7071	-3528.1149	2403.4920
CAPTAINCY EXP	229322.1609	116416.4233	1.9698	0.0522	-2184.6640	460828.9857
RUNS-S	142.5700	156.5644	0.9106	0.3651	-168.7755	453.9154
HIGH SCORE	-7143.1380	2364.3774	-3.0211	0.0033	-11844.9618	-2441.3143
AVE	26776.0292	7121.6081	3.7598	0.0003	12613.9304	40938.1279
SR -B	-1295.2083	1252.3783	-1.0342	0.3040	-3785.7000	1195.2833
SIXERS	5036.2248	4008.5624	1.2564	0.2125	-2935.2412	13007.6908
RUNS-C	28.7150	261.2174	0.1099	0.9127	-490.7445	548.1745
WKTS	4615.6867	6748.8806	0.6839	0.4959	-8805.2026	18036.5760
AVE-BL	4683.1892	10003.4036	0.4682	0.6409	-15209.6758	24576.0542
ECON	-5279.1917	11949.8517	-0.4418	0.6598	-29042.7823	18484.3990
SR-BL	-2714.9916	13715.5379	-0.1980	0.8436	-29989.8429	24559.8598

Omnibus:	4.738	Durbin-Watson:	2.112
Prob(Omnibus):	0.094	Jarque-Bera (JB):	4.802
Skew:	0.509	Prob(JB):	0.091
Kurtosis:	2.730	Condition No.:	19855

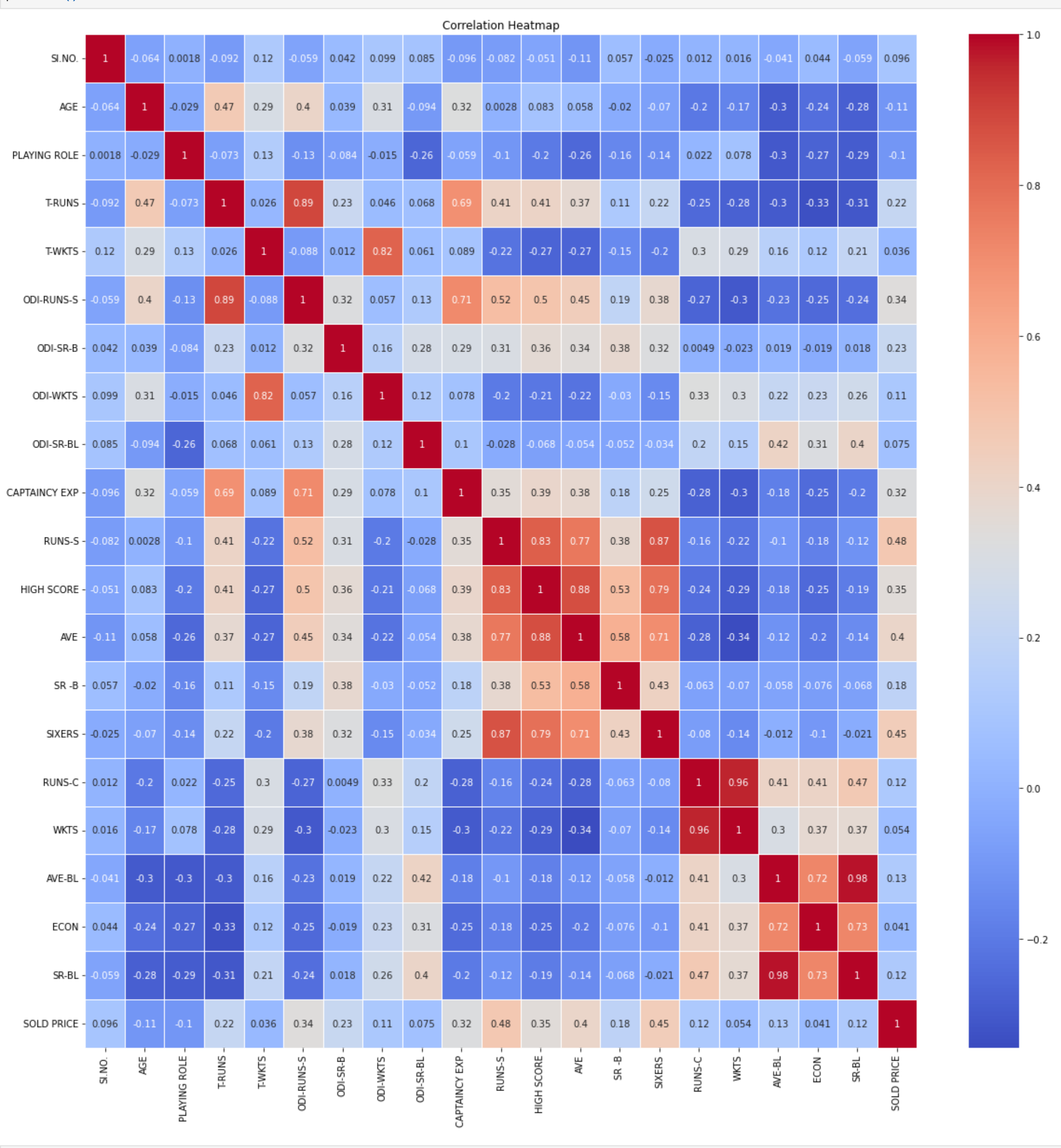
In [49]:
```
correlation_matrix = df.corr()
correlation_matrix
```

Out[49]:

	Sl.NO.	AGE	PLAYING ROLE	T-RUNS	T-WKTS	ODI-RUNS-S	ODI-SR-B	ODI-WKTS	ODI-SR-BL	CAPTAINCY EXP	...	HIGH SCORE	AVE	SR -B	SIXERS	RUNS-C	WKTS	AVE-BL	ECON
Sl.NO.	1.000000	-0.064235	0.001824	-0.091722	0.121835	-0.059029	0.042439	0.098777	0.084678	-0.096388	...	-0.051490	-0.108076	0.057132	-0.024863	0.011956	0.016440	-0.040547	0.044308
AGE	-0.064235	1.000000	-0.028686	0.469402	0.293473	0.403131	0.038531	0.306949	-0.093574	0.323228	...	0.083028	0.057569	-0.020303	-0.070133	-0.203835	-0.170092	-0.300514	-0.236669
PLAYING ROLE	0.001824	-0.028686	1.000000	-0.072742	0.134025	-0.129866	-0.084269	-0.014578	-0.262087	-0.059056	...	-0.199238	-0.255866	-0.159495	-0.136330	0.021715	0.078337	-0.303585	-0.272129
T-RUNS	-0.091722	0.469402	-0.072742	1.000000	0.026285	0.892823	0.231411	0.045505	0.067700	0.690647	...	0.411209	0.374046	0.114298	0.216571	-0.253083	-0.277157	-0.298999	-0.329022
T-WKTS	0.121835	0.293473	0.134025	0.026285	1.000000	-0.088276	0.012052	0.822940	0.060641	0.088782	...	-0.268432	-0.265540	-0.147752	-0.198036	0.297302	0.289735	0.162456	0.117530
ODI-RUNS-S	-0.059029	0.403131	-0.129866	0.892823	-0.088276	1.000000	0.319264	0.056554	0.126795	0.714058	...	0.495765	0.446280	0.194111	0.376012	-0.268950	-0.302745	-0.225109	-0.249795
ODI-SR-B	0.042439	0.038531	-0.084269	0.231411	0.012052	0.319264	1.000000	0.160114	0.284584	0.291373	...	0.360086	0.340327	0.375371	0.320041	0.004855	-0.022502	0.018958	-0.019067
ODI-WKTS	0.098777	0.306949	-0.014578	0.045505	0.822940	0.056554	0.160114	1.000000	0.124361	0.077536	...	-0.209109	-0.222062	-0.030404	-0.148722	0.327122	0.302466	0.217302	0.230221
ODI-SR-BL	0.084678	-0.093574	-0.262087	0.067700	0.060641	0.126795	0.284584	0.124361	1.000000	0.103315	...	-0.067615	-0.053766	-0.051684	-0.033616	0.196887	0.151594	0.416627	0.313295
CAPTAINCY EXP	-0.096388	0.323228	-0.059056	0.690647	0.088782	0.714058	0.291373	0.077536	0.103315	1.000000	...	0.392749	0.378271	0.176919	0.254991	-0.277903	-0.300074	-0.183775	-0.248974
RUNS-S	-0.082366	0.002776	-0.102518	0.410143	-0.218544	0.523955	0.306338	-0.198101	-0.027772	0.352422	...	0.834631	0.767023	0.380022	0.866213	-0.160762	-0.220959	-0.101818	-0.179459
HIGH SCORE	-0.051490	0.083028	-0.199238	0.411209	-0.268432	0.495765	0.360086	-0.209109	-0.067615	0.392749	...	1.000000	0.876249	0.531028	0.788439	-0.235327	-0.291494	-0.176049	-0.254494
AVE	-0.108076	0.057569	-0.255866	0.374046	-0.265540	0.446280	0.340327	-0.222062	-0.053766	0.378271	...	0.876249	1.000000	0.583579	0.705365	-0.278482	-0.343958	-0.120430	-0.202301
SR -B	0.057132	-0.020303	-0.159495	0.114298	-0.147752	0.194111	0.375371	-0.030404	-0.051684	0.176919	...	0.531028	0.583579	1.000000	0.425394	-0.063179	-0.069823	-0.057895	-0.076228
SIXERS	-0.024863	-0.070133	-0.136330	0.216571	-0.198036	0.376012	0.320041	-0.148722	-0.033616	0.254991	...	0.788439	0.705365	0.425394	1.000000	-0.080295	-0.144558	-0.012210	-0.101027
RUNS-C	0.011956	-0.203835	0.021715	-0.253083	0.297302	-0.268950	0.004855	0.327122	0.196887	-0.277903	...	-0.235327	-0.278482	-0.063179	-0.080295	1.000000	0.959210	0.409505	0.410502
WKTS	0.016440	-0.170092	0.078337	-0.277157	0.289735	-0.302745	-0.022502	0.302466	0.151594	-0.300074	...	-0.291494	-0.343958	-0.069823	-0.144558	0.959210	1.000000	0.297822	0.374875
AVE-BL	-0.040547	-0.300514	-0.303585	-0.298999	0.162456	-0.225109	0.018958	0.217302	0.416627	-0.183775	...	-0.176049	-0.120430	-0.057895	-0.012210	0.409505	0.297822	1.000000	0.722758
ECON	0.044308	-0.236669	-0.272129	-0.329022	0.117530	-0.249795	-0.019067	0.230221	0.313295	-0.248974	...	-0.254494	-0.202301	-0.076228	-0.101027	0.410502	0.374875	0.722758	1.000000
SR-BL	-0.058574	-0.282870	-0.293917	-0.309105	0.205208	-0.237517	0.017522	0.259800	0.402495	-0.198688	...	-0.192990	-0.140809	-0.068248	-0.021412	0.471224	0.368264	0.983415	0.726480
SOLD PRICE	0.096150	-0.106201	-0.103657	0.216752	0.035767	0.337834	0.226880	0.112327	0.075408	0.318835	...	0.347473	0.396519	0.184278	0.450609	0.116476	0.053796	0.128406	0.040679

21 rows × 21 columns

In [50]:
```
plt.figure(figsize=(20,20))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')


plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

The data under the filename “BANK MARKETING” provides data on a bank study to classify the customers of the whether a loan can be given or not, depending on various attributes of the customer. Write the Python Programming to

a. Build a logistic Regression, SVM and Random Forest Models “Loan Approval” of the customer based on all the independent variables b. Compare 3 models according to accuracy b. Plot the confusion matrix, classification report and accuracy.

In [3]:
```
df = pd.read_csv("/Users/shubham/Desktop/Assigment/Bank Marketing.csv")
```

In [7]:
```
df.head(20)
```

Out[7]:

	S.No	age	job	BlueCollar	SelfEmployed	Management	Unemployed	marital	Divorced	Married	...	default	CreditDefault	balance	housing	HousingLoan	loan	PersonalLoan	campaign	previous
0	1	59	unemployed	0	0	0	1	married	0	1	...	no	0	0	no	0	no	0	2	0
1	2	36	Others	0	1	0	0	married	0	1	...	no	0	3057	no	0	no	0	4	0
2	3	47	blue-collar	1	0	0	0	divorced	1	0	...	no	0	126	yes	1	no	0	2	0
3	4	43	management	0	0	1	0	divorced	1	0	...	no	0	388	yes	1	no	0	2	0
4	5	34	self-employed	0	1	0	0	single	0	0	...	no	0	462	no	0	no	0	3	0
5	6	49	Others	0	0	0	0	married	0	1	...	no	0	320	no	0	no	0	4	0
6	7	54	Others	0	0	0	0	married	0	1	...	no	0	-315	no	0	yes	1	1	0
7	8	29	Others	0	0	0	0	single	0	0	...	no	0	778	yes	1	no	0	2	0
8	9	40	management	0	0	1	0	married	0	1	...	no	0	-1212	yes	1	no	0	2	18
9	10	32	Others	0	0	0	0	single	0	0	...	no	0	678	yes	1	no	0	1	4
10	11	40	management	0	0	1	0	married	0	1	...	no	0	542	yes	1	no	0	1	0
11	12	56	Others	0	0	0	0	divorced	1	0	...	yes	1	1238	no	0	no	0	1	0
12	13	40	management	0	0	1	0	single	0	0	...	no	0	1361	no	0	no	0	1	0
13	14	58	management	0	0	1	0	married	0	1	...	no	0	1831	no	0	no	0	1	0
14	15	29	management	0	0	1	0	married	0	1	...	no	0	318	yes	1	no	0	2	4
15	16	37	Others	0	0	0	0	single	0	0	...	no	0	228	yes	1	no	0	2	0
16	17	31	Others	0	0	0	0	single	0	0	...	no	0	433	no	0	no	0	2	0
17	18	35	unemployed	0	0	0	1	married	0	1	...	no	0	978	no	0	no	0	3	3
18	19	44	Others	0	0	0	0	single	0	0	...	no	0	244	yes	1	no	0	4	0
19	20	26	unemployed	0	0	0	1	single	0	0	...	no	0	622	no	0	no	0	2	0

20 rows × 24 columns

In [11]:
```
import pandas as pd

df = pd.read_csv("/Users/shubham/Desktop/Assigment/Bank Marketing.csv")

categorical_columns = ['job', 'marital', 'default', 'housing', 'loan', 'Loan_Approval','education']

df_encoded = pd.get_dummies(df, columns=categorical_columns)
```

In [12]:
```
df_encoded
```

Out[12]:

	S.No	age	BlueCollar	SelfEmployed	Management	Unemployed	Divorced	Married	Primary	Secondary	...	housing_no	housing_yes	loan_no	loan_yes	Loan_Approval_no	Loan_Approval_yes	education_
0	1	59	0	0	0	1	0	1	1	0	...	1	0	1	0	1	0	
1	2	36	0	1	0	0	0	1	0	0	...	1	0	1	0	0	1	
2	3	47	1	0	0	0	1	0	1	0	...	0	1	1	0	0	1	
3	4	43	0	0	1	0	1	0	0	0	...	0	1	1	0	0	1	
4	5	34	0	1	0	0	0	0	0	0	...	1	0	1	0	0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4516	4517	58	1	0	0	0	0	1	1	0	...	0	1	1	0	1	0	
4517	4518	38	0	0	0	0	0	1	1	0	...	0	1	1	0	1	0	
4518	4519	53	0	1	0	0	0	1	0	0	...	0	1	0	1	1	0	
4519	4520	49	1	0	0	0	0	1	1	0	...	0	1	1	0	1	0	
4520	4521	35	1	0	0	0	0	1	0	1	...	0	1	0	1	1	0	

4521 rows × 37 columns

In [14]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
```

In [23]:
```
target_column = 'Loan_Approval'

X = df.drop(columns=[target_column])
y = df[target_column]


categorical_columns = X.select_dtypes(include=['object']).columns
numerical_columns = X.select_dtypes(exclude=['object']).columns

X = pd.get_dummies(X, columns=categorical_columns)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


scaler = StandardScaler()
X_train[numerical_columns] = scaler.fit_transform(X_train[numerical_columns])
X_test[numerical_columns] = scaler.transform(X_test[numerical_columns])
```

In [27]:
```
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)
```

Out[27]:
```
LogisticRegression(random_state=42)
```

In [25]:
```
svm_model = SVC(random_state=42)
svm_model.fit(X_train, y_train)
```

Out[25]:
```
SVC(random_state=42)
```

In [26]:
```
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

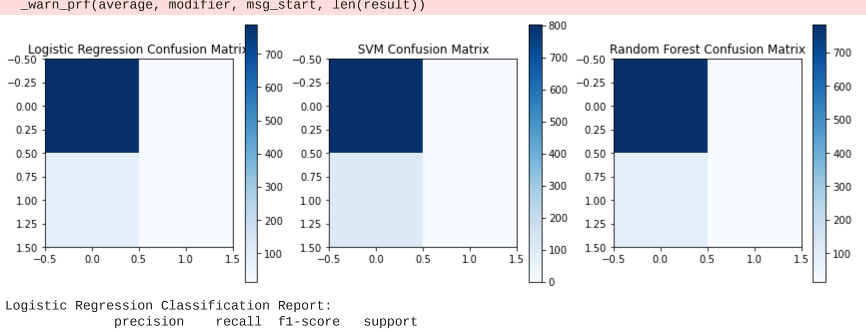
Out[26]:
```
RandomForestClassifier(random_state=42)
```

In [29]:
```
y_pred_lr = lr_model.predict(X_test)
y_pred_svm = svm_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)


accuracy_lr = accuracy_score(y_test, y_pred_lr)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

print(f"Logistic Regression Accuracy: {accuracy_lr:.2f}")
print(f"SVM Accuracy: {accuracy_svm:.2f}")
print(f"Random Forest Accuracy: {accuracy_rf:.2f}")
```

```
Logistic Regression Accuracy: 0.89
SVM Accuracy: 0.89
Random Forest Accuracy: 0.88
```

In [30]:
```
cm_lr = confusion_matrix(y_test, y_pred_lr)
cm_svm = confusion_matrix(y_test, y_pred_svm)
cm_rf = confusion_matrix(y_test, y_pred_rf)

report_lr = classification_report(y_test, y_pred_lr)
report_svm = classification_report(y_test, y_pred_svm)
report_rf = classification_report(y_test, y_pred_rf)


plt.figure(figsize=(12, 4))

plt.subplot(131)
plt.title("Logistic Regression Confusion Matrix")
plt.imshow(cm_lr, cmap='Blues', interpolation='nearest')
plt.colorbar()

plt.subplot(132)
plt.title("SVM Confusion Matrix")
plt.imshow(cm_svm, cmap='Blues', interpolation='nearest')
plt.colorbar()

plt.subplot(133)
plt.title("Random Forest Confusion Matrix")
plt.imshow(cm_rf, cmap='Blues', interpolation='nearest')
plt.colorbar()

plt.tight_layout()
plt.show()


print("Logistic Regression Classification Report:")
print(report_lr)

print("SVM Classification Report:")
print(report_svm)

print("Random Forest Classification Report:")
print(report_rf)
```

```
/Users/shubham/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/shubham/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/shubham/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and bein
g set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```




```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

          no       0.90      0.98      0.94       803
         yes       0.59      0.19      0.28       102

    accuracy                           0.89       905
   macro avg       0.75      0.59      0.61       905
weighted avg       0.87      0.89      0.87       905

SVM Classification Report:
              precision    recall  f1-score   support

          no       0.89      1.00      0.94       803
         yes       0.00      0.00      0.00       102

    accuracy                           0.89       905
   macro avg       0.44      0.50      0.47       905
weighted avg       0.79      0.89      0.83       905

Random Forest Classification Report:
              precision    recall  f1-score   support

          no       0.90      0.98      0.94       803
         yes       0.42      0.14      0.21       102

    accuracy                           0.88       905
   macro avg       0.66      0.56      0.57       905
weighted avg       0.85      0.88      0.85       905
```

Logistic Regression:

Accuracy: 0.89 Precision (yes class): 0.59 Recall (yes class): 0.19 F1-score (yes class): 0.28 SVM:

Accuracy: 0.89 Precision (yes class): 0.00 Recall (yes class): 0.00 F1-score (yes class): 0.00 Random Forest:

Accuracy: 0.88 Precision (yes class): 0.42 Recall (yes class): 0.14 F1-score (yes class): 0.21 Based on these metrics, it seems that Logistic Regression performs the best among the three models. It has the highest accuracy and the highest F1-score for the "yes" class, indicating a better balance between precision and recall for that class.

In [ ]: