Our model today is about bank Loan Approval prediction for buying a property, As we know alot of things needs to be verified before someone is accepted to get a loan due to several reasons in which one of them is that he might not be able to pay back the loan.

The dataset we will use consists of the following features:

1. Loan_ID: Id of the customer
2. Gender: Male or Female
3. Married: Married or Single
4. Dependents: Number of people that depends on the customer applying for the loan
5. Education: Whether the customer is a graduate or not
6. Self_Employed: Whether the customer works by himself or employed by a company
7. ApplicantIncome: Income of the customer applying for the loan
8. CoapplicantIncome: Coapplicant income
9. LoanAmount: The amount to be loaned by the customer
10. Loan_Amount_Term: The duration in which the customer is supposed to pay back the loan
11. Credit_History: Whether the customer fullfilled the terms for the loan
12. Property_Area: The place in which the customer will buy his property
13. Loan_Status: Customer status for the approval of the loan

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler

import warnings
warnings.filterwarnings("ignore")

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import TomekLinks
from collections import Counter
from imblearn.over_sampling import SMOTE

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.metrics import accuracy_score,recall_score,precision_score,f1_score,confusion
```

Reading Data

In [2]:

```
df = pd.read_csv("/Users/HP/Desktop/loan_Data.csv")
df
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loar
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loar
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	
...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	

614 rows × 13 columns

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [4]:

```
df.describe()
```

Out[4]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

Checking for nulls & duplicates

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

- Our data is 614 rows, we found out that there are some missing values with the maximum of a feature is 50 which is (Credit_history)
- Some of the features having null values are categorical features which we can not replace it with the mean such as the gender , Self_employed and married,
- Some of of the features having null values are numerical features but it either 1 or 0 such as Credit_History.
- Decided to replace the nulls of the married, gender, Self_Employed, Loan_Amount_Term, Dependents and Credit_History by the mode as our dataset size is small and we don't want to lose any more data.
- Decided to replace the LoanAmount column by it's mean.

In [6]:

```
df['Credit_History'].fillna(value = df['Credit_History'].mode()[0], inplace = True)
df['Married'].fillna(value = df['Married'].mode()[0], inplace = True)
df['Gender'].fillna(value = df['Gender'].mode()[0], inplace = True)
df['Dependents'].fillna(value = df['Dependents'].mode()[0], inplace = True)
df['Self_Employed'].fillna(value = df['Self_Employed'].mode()[0], inplace = True)
df['Loan_Amount_Term'].fillna(value = df['Loan_Amount_Term'].mode()[0], inplace = True)
df['LoanAmount'].fillna(value = df['LoanAmount'].mean(), inplace = True)
```

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [8]:

```
df.duplicated().sum()
```

Out[8]:

```
0
```

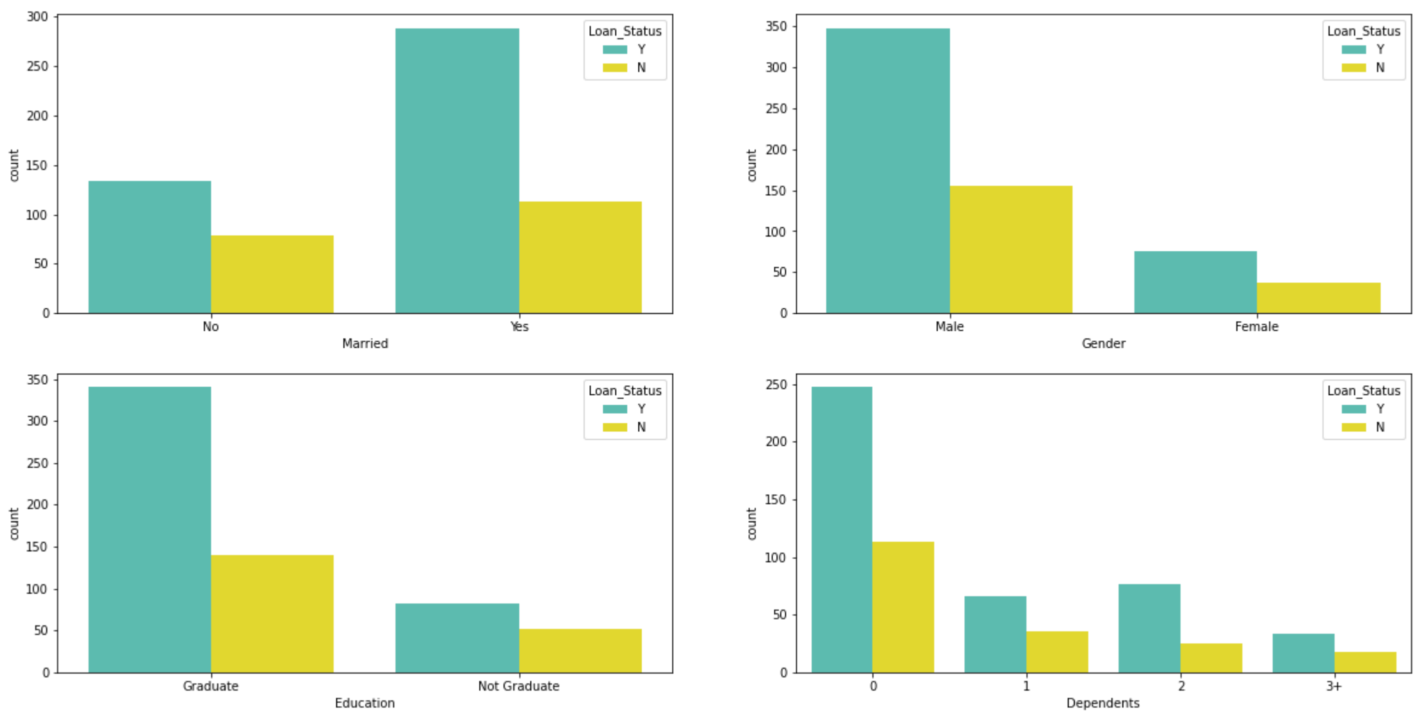
EDA

In [9]:

```
plt.figure(figsize = (20,10))

# subplot 1
plt.subplot(2, 2, 1)
sns.countplot(x='Married', hue = 'Loan_Status',data = df , palette = ["#4ccbbb", "#fff111'

# subplot 2
plt.subplot(2, 2, 2)
sns.countplot(x = 'Gender', hue = 'Loan_Status',data = df, palette = ["#4ccbbb", "#fff111'

# subplot 3
plt.subplot(2, 2, 3)
sns.countplot(x = 'Education', hue = 'Loan_Status',data = df, palette = ["#4ccbbb", "#fff1

# subplot 4
plt.subplot(2, 2, 4)
sns.countplot(x = 'Dependents', hue = 'Loan_Status',data = df, palette = ["#4ccbbb", "#ff1
```

Out[9]:

```
<Axes: xlabel='Dependents', ylabel='count'>
```



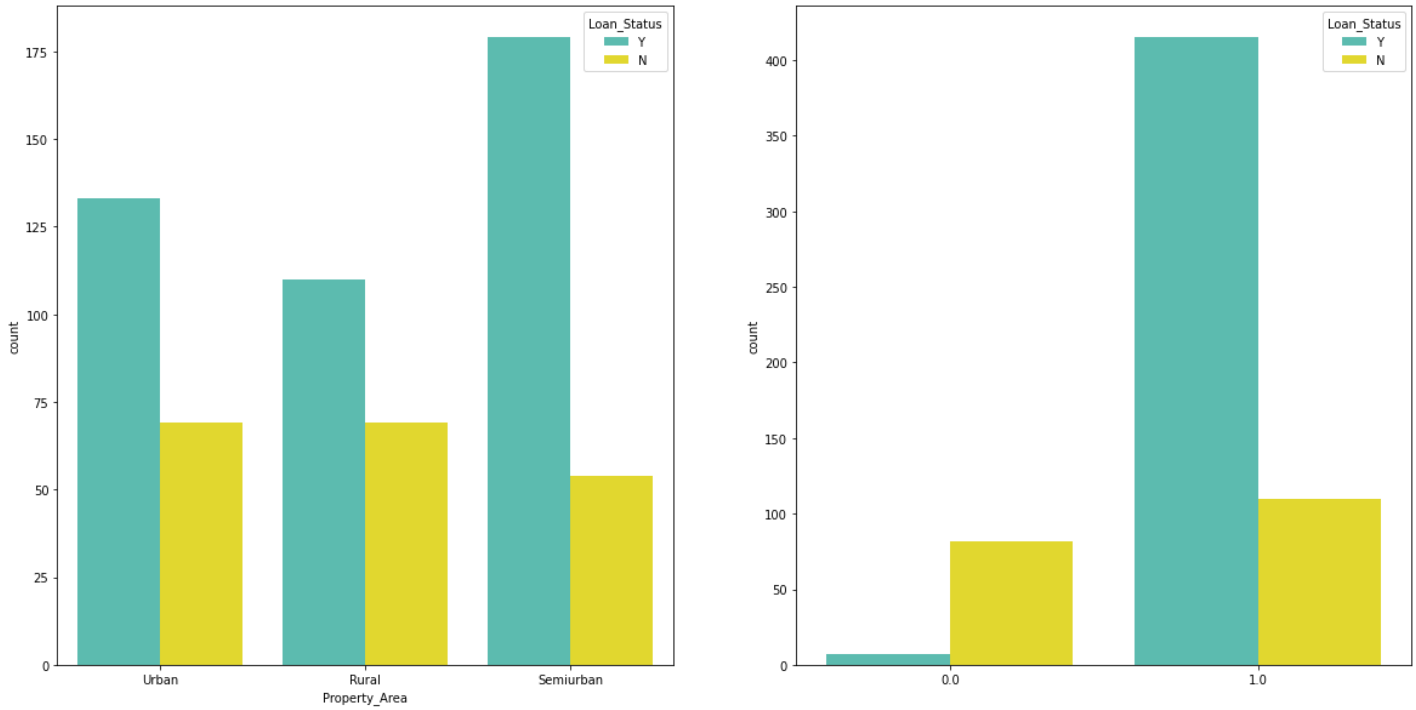
- We find out that married people has higher chance to be approved for a loan that non-married people
- Males has higher chances for loan approval are higher than female chances
- Graduated people has higher percentage of acceptance for loan approval than not graduate
- The more the dependency that a person has, the less chances he will be accepted for a loan approval

In [10]:

```
plt.figure(figsize = (20,10))

# subplot 1
plt.subplot(1, 2, 1)
sns.countplot(x='Property_Area', hue = 'Loan_Status',data = df , palette = ["#4ccbbb", "#1

# subplot 2
plt.subplot(1, 2, 2)
sns.countplot(x= df['Credit_History'].values, hue = 'Loan_Status',data = df , palette = ['
```

```
<Axes: ylabel='count'>
```

Out[10]:

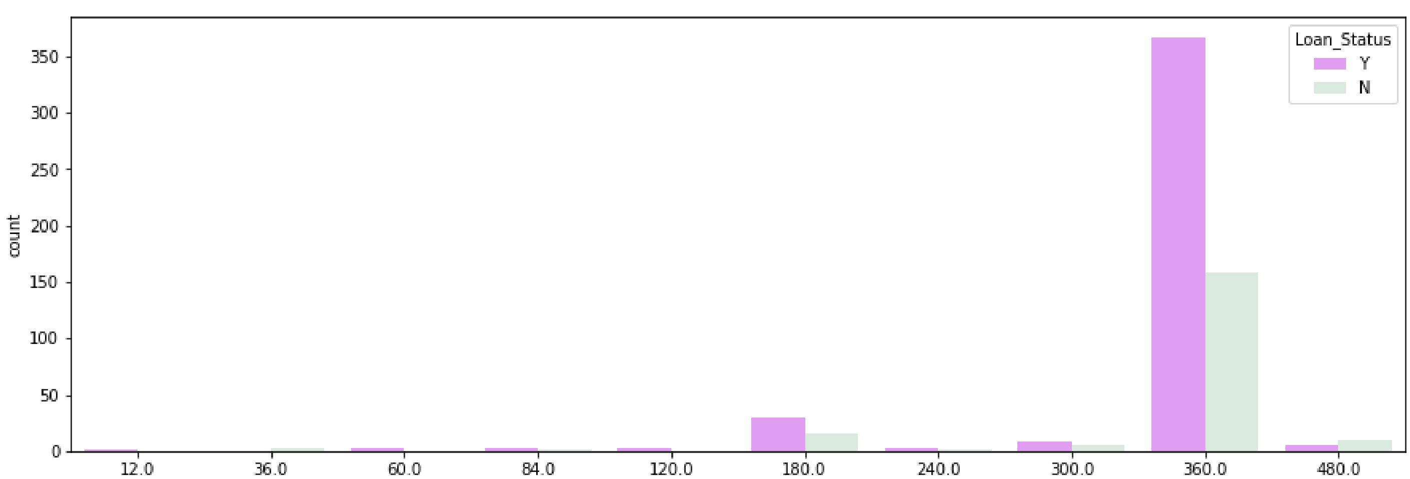


- If a person credit history is positive (= 1), the chances of getting approved for a loan is approximately 35 times higher compared to a person in which his credit history is 0 (didn't meet the guidelines)
- The property area doesn't have that much affect on a person chance for a loan approval. No matter where a person lives, he has good chances for a loan approval

In [11]:

```
plt.figure(figsize = (15,5))
sns.countplot(x= df['Loan_Amount_Term'].values, hue = 'Loan_Status',data = df , palette=['
```

Out[11]: <Axes: ylabel='count'>

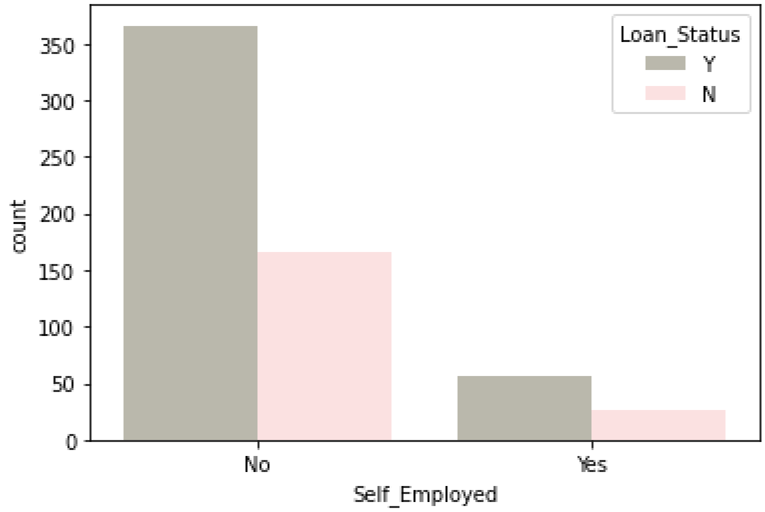


- A person has a very great chance to be accpeted for a loan approval if the loan term is a year compared to other loan terms.
- followed by, is if the loan term is half a year.
- The rest of the loan terms, the chances for a loan approval seems to be impossible.

In [12]:

```
sns.countplot(x='Self_Employed', hue = 'Loan_Status',data = df , palette=["#bcbaaa", "#ffc
```

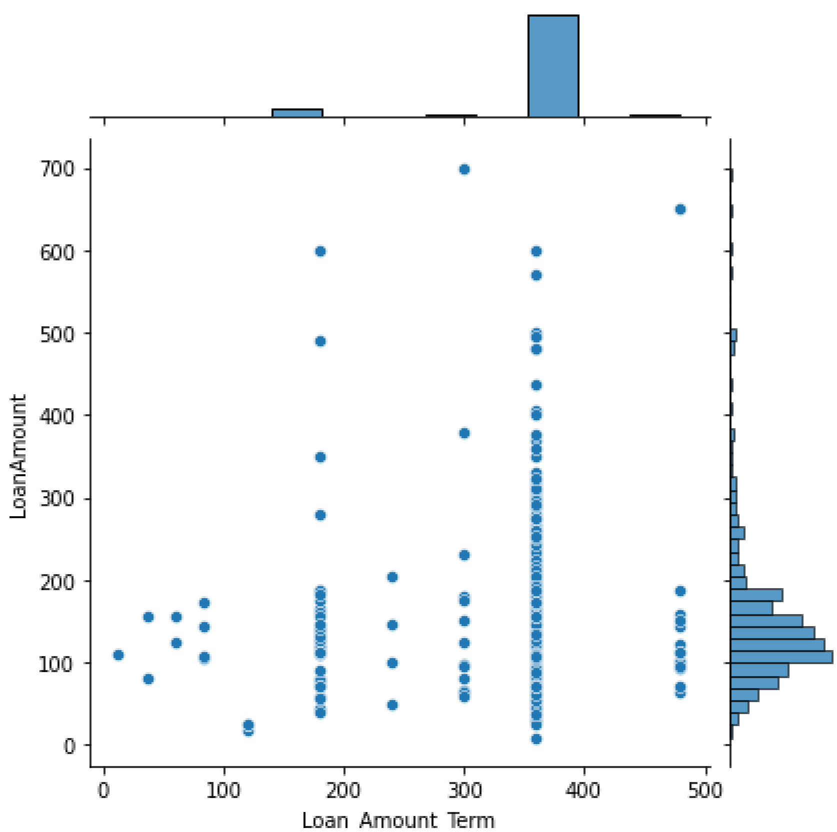
Out[12]: <Axes: xlabel='Self_Employed', ylabel='count'>



- There is higher chance of getting a loan approval if the person is not self employed than being self-employed

In [13]:
```
sns.jointplot(x = 'Loan_Amount_Term', y = 'LoanAmount', data = df, kind ='scatter')
```

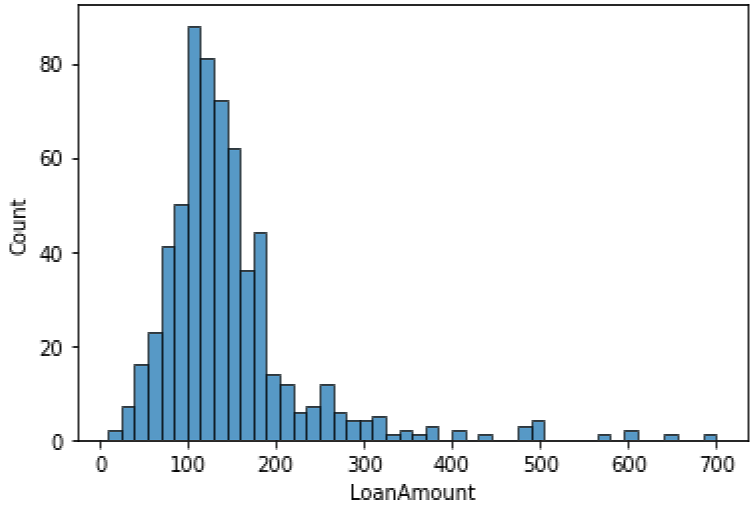
Out[13]: <seaborn.axisgrid.JointGrid at 0x2d2f68ba8e0>



- We can find that the majority of the loan duration is a year followed by a half year
- The higher the amount the amount to be loaned, the higher the chance to be accepted for a bigger loan duration.

In [14]:
```
sns.histplot(x = 'LoanAmount', data = df)
```

Out[14]: <Axes: xlabel='LoanAmount', ylabel='Count'>



- Majority of the Loan Amounts is between 80 to 160

Encoding of Categorical Features

In [15]:

```
df.replace({'Married':{'No':0 ,'Yes':1}, 'Gender':{'Female':0 ,'Male':1}, 'Education':{'No
            'Self_Employed':{'No':0 ,'Yes':1}, 'Loan_Status':{'N':0 ,'Y':1}}, inplace = Tru
```

- We replaced the categories having two labels only with 0 and 1
- the rest of the categories to be encoded will be dealt with one hot encoding as they have multi-labels (more than 2 labels)

In [16]:

```
df
```

Out[16]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	Loar
0	LP001002	1	0	0	1	0	5849	0.0	14
1	LP001003	1	1	1	1	0	4583	1508.0	12
2	LP001005	1	1	0	1	1	3000	0.0	6
3	LP001006	1	1	0	0	0	2583	2358.0	12
4	LP001008	1	0	0	1	0	6000	0.0	14
...	...	...	...	...	...	...	...	...	
609	LP002978	0	0	0	1	0	2900	0.0	7
610	LP002979	1	1	3+	1	0	4106	0.0	4
611	LP002983	1	1	1	1	0	8072	240.0	25
612	LP002984	1	1	2	1	0	7583	0.0	18
613	LP002990	0	0	0	1	1	4583	0.0	13

614 rows × 13 columns

In [17]:

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

Out[17]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	l
0	1	0	0	1	0	5849	0.0	146.412162	
1	1	1	1	1	0	4583	1508.0	128.000000	
2	1	1	0	1	1	3000	0.0	66.000000	
3	1	1	0	0	0	2583	2358.0	120.000000	
4	1	0	0	1	0	6000	0.0	141.000000	
...	...	...	...	...	...	...	...	...	
609	0	0	0	1	0	2900	0.0	71.000000	
610	1	1	3+	1	0	4106	0.0	40.000000	
611	1	1	1	1	0	8072	240.0	253.000000	
612	1	1	2	1	0	7583	0.0	187.000000	
613	0	0	0	1	1	4583	0.0	133.000000	

614 rows × 12 columns

- Dropped out the Loan_ID column as there no use of it in our model

In [18]:

```
pd.get_dummies(df['Dependents'], drop_first = False)
```

Out[18]:

	0	1	2	3+
0	1	0	0	0
1	0	1	0	0
2	1	0	0	0
3	1	0	0	0
4	1	0	0	0
...	...	...	...	...
609	1	0	0	0
610	0	0	0	1
611	0	1	0	0
612	0	0	1	0
613	1	0	0	0

614 rows × 4 columns

- This is a nomial feature which is best to be dealt by one hot encoding.

In [19]:

```
df = df.join(pd.get_dummies(df['Dependents'], drop_first = False, prefix = 'dependent')).c
df
```

Out[19]:

	Gender	Married	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	1	0	1	0	5849	0.0	146.412162	

	Gender	Married	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
1	1	1	1	0	4583	1508.0	128.000000	
2	1	1	1	1	3000	0.0	66.000000	
3	1	1	0	0	2583	2358.0	120.000000	
4	1	0	1	0	6000	0.0	141.000000	
...	...	...	...	...	...	...	...	
609	0	0	1	0	2900	0.0	71.000000	
610	1	1	1	0	4106	0.0	40.000000	
611	1	1	1	0	8072	240.0	253.000000	
612	1	1	1	0	7583	0.0	187.000000	
613	0	0	1	1	4583	0.0	133.000000	

614 rows × 15 columns

In [20]:

```
pd.get_dummies(df['Property_Area'], drop_first = False, prefix = 'property')
```

Out[20]:

	property_Rural	property_Semiurban	property_Urban
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
609	1	0	0
610	1	0	0
611	0	0	1
612	0	0	1
613	0	1	0

614 rows × 3 columns

- This is a nomial feature which is best to be dealt by one hot encoding.

In [21]:

```
df = df.join(pd.get_dummies(df['Property_Area'], drop_first = False, prefix = 'property'))
df
```

Out[21]:

	Gender	Married	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	1	0	1	0	5849	0.0	146.412162	
1	1	1	1	0	4583	1508.0	128.000000	
2	1	1	1	1	3000	0.0	66.000000	

	Gender	Married	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
3	1	1	0	0	2583	2358.0	120.000000	
4	1	0	1	0	6000	0.0	141.000000	
...	...	...	...	...	...	...	...	
609	0	0	1	0	2900	0.0	71.000000	
610	1	1	1	0	4106	0.0	40.000000	
611	1	1	1	0	8072	240.0	253.000000	
612	1	1	1	0	7583	0.0	187.000000	
613	0	0	1	1	4583	0.0	133.000000	

614 rows × 17 columns

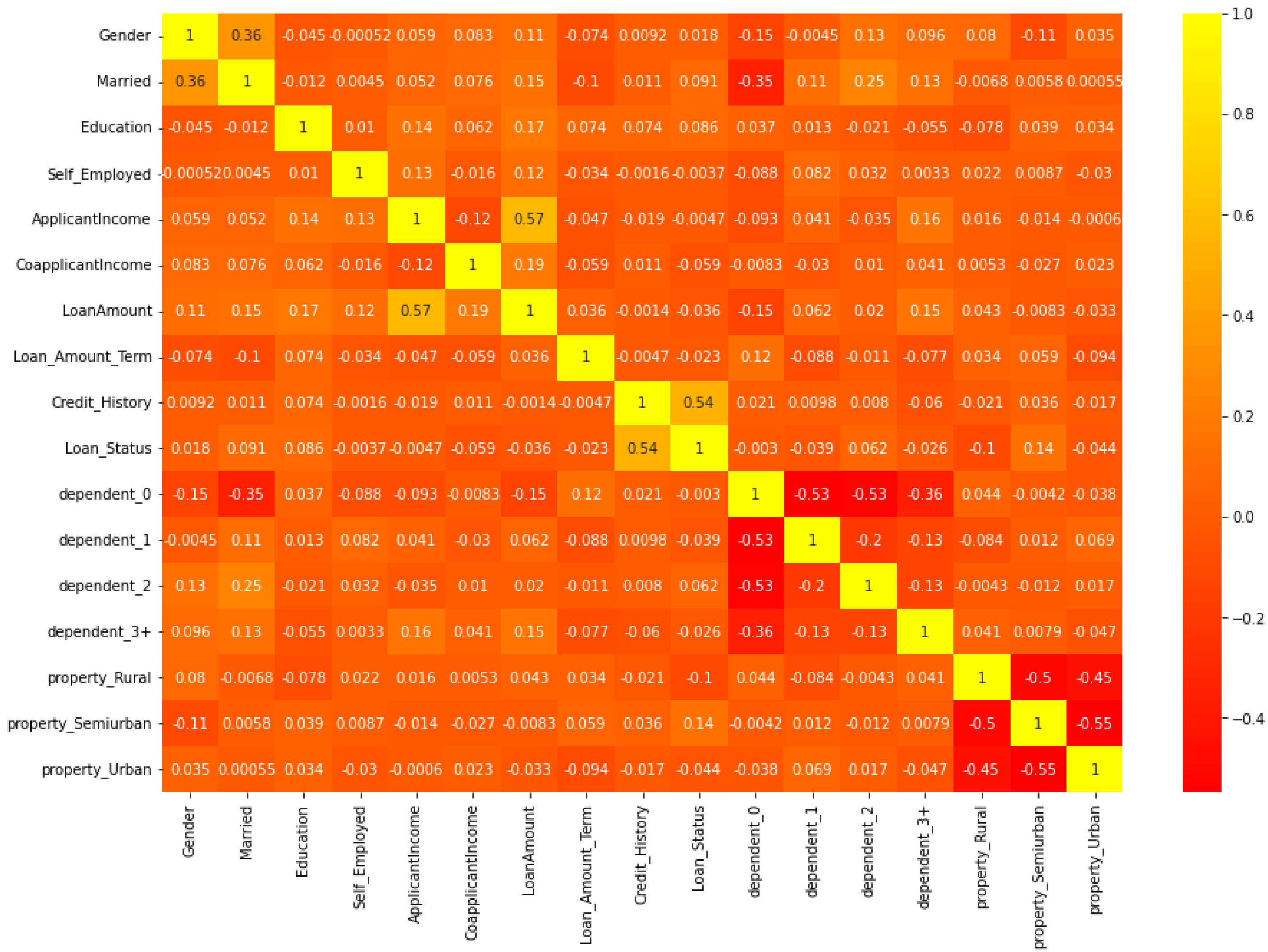
Correlation

In [22]:

```
corr = df.corr()
plt.figure( figsize = (15,10))
sns.heatmap(corr , annot = True, cmap = 'autumn')
```

Out[22]:

```
<Axes: >
```



- We can find that there is no features that is heavily correlated to each other
- This is great for our performance of the model
- No feature selection is needed

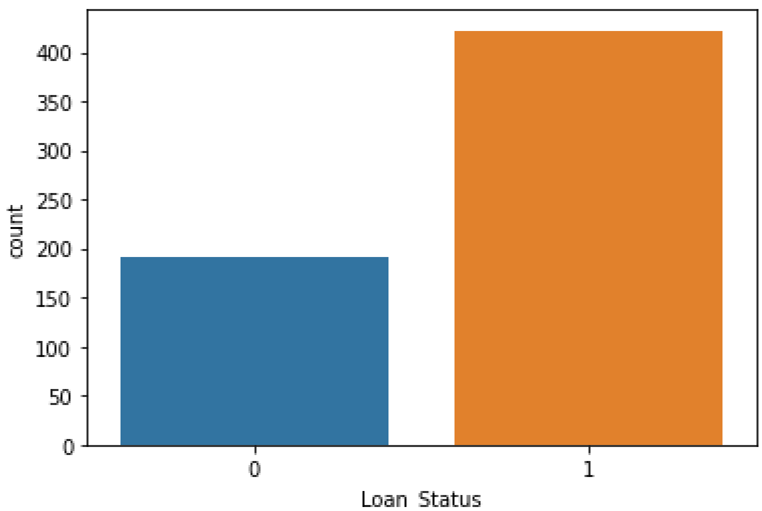
Splitting of data

In [23]:
```
x = df.drop(['Loan_Status'], axis = 1)
y = df['Loan_Status']
```

Checking for imbalanced data

In [24]:
```
sns.countplot(df['Loan_Status'])
df['Loan_Status'].value_counts()
```

Out[24]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```



- We find out there is huge difference between yes and no labels which is about 50 %
- OverSampling will be the technique used to solve the problem of the imbalanced of data so that our model will make good prediction

In [25]:
```
Counter(y)
```

Out[25]:
```
Counter({1: 422, 0: 192})
```

In [26]:
```
smote = SMOTE(k_neighbors = 5)
x, y = smote.fit_resample(x,y)
```

In [27]:
```
Counter(y)
```

Out[27]:
```
Counter({1: 422, 0: 422})
```

- Now we are ready to train test split

train test split

In [28]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3, random_state =42

x_train,x_val,y_train,y_val = train_test_split(x_train,y_train,test_size = 0.2, random_sta
```

Scaling

In [29]:

```
scaler = StandardScaler()

x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
x_test = scaler.transform(x_test)
```

- We just scaled the the input features
- We applied a fit_transform for the x_train
- For the rest, we just applied a transform function for them (x_val, x_test)

KNN

In [30]:

```
# Use grid search to find best value
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

params_grid = {
    'n_neighbors':[3,4,5,6,7,8,9,10]
}

grid = GridSearchCV(
    knn,
    params_grid,
    cv = 5
)
grid.fit(x_train,y_train)

print(f'the best value of k = {grid.best_params_}')
```

```
the best value of k = {'n_neighbors': 10}
```

In [31]:

```
## check overfitting

knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(x_train,y_train)

x_train_pred = knn.predict(x_train)
train_score = accuracy_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = knn.predict(x_val)
val_score = accuracy_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')

# scores are equal so no overfitting
```

```
the train score is =0.8771186440677966
the valid score is =0.8305084745762712
```

In [32]:

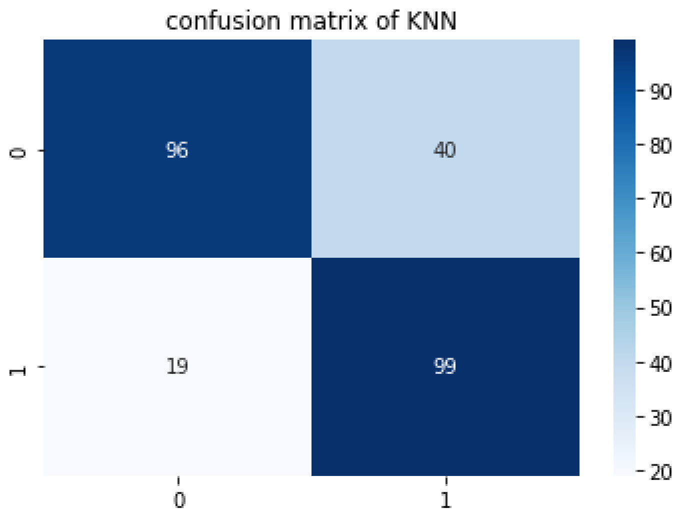
```python
knn = KNeighborsClassifier(n_neighbors = 3)

kn = knn.fit(x_train,y_train)
y_pred = kn.predict(x_test)
```

In [33]:

```python
cnf_mat = confusion_matrix(y_test,y_pred)
sns.heatmap( cnf_mat , annot = True , cmap = 'Blues')
plt.title('confusion matrix of KNN')
```

Out[33]:

```
Text(0.5, 1.0, 'confusion matrix of KNN')
```



In [34]:

```python
accuracy_knn = accuracy_score(y_test, y_pred)
print(f'the accuracy of the Knn model is = {accuracy_knn * 100} %')
recall = recall_score(y_test, y_pred)
print(f'the recall of the Knn model is = {recall * 100} %')
precision = precision_score(y_test, y_pred)
print(f'the precision of the Knn model is = {precision * 100} %')
f1 = f1_score(y_test, y_pred)
print(f'the f1_score of the Knn model is = {f1 * 100} %')
```

```
the accuracy of the Knn model is = 76.77165354330708 %
the recall of the Knn model is = 83.89830508474576 %
the precision of the Knn model is = 71.22302158273382 %
the f1_score of the Knn model is = 77.04280155642024 %
```

In [35]:

```python
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.83      0.71      0.76       136
           1       0.71      0.84      0.77       118

    accuracy                           0.77       254
   macro avg       0.77      0.77      0.77       254
weighted avg       0.78      0.77      0.77       254
```

Random Forest

In [36]:
```python
# Use grid search to find best value
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

params_grid = {
    'max_depth':[3,4,5,6,7,8,9,10],
    'criterion': ['gini', 'entropy', 'log_loss']
}

grid = GridSearchCV(
    rf,
    params_grid,
    cv = 5
)
grid.fit(x_train,y_train)

print(f'the best value is = {grid.best_params_}')
```

```
the best value is = {'criterion': 'gini', 'max_depth': 8}
```

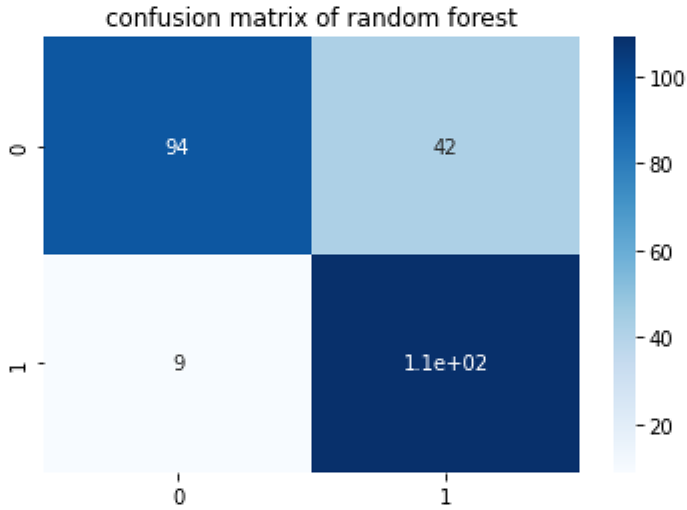
In [41]:
```python
## check overfitting

rf = RandomForestClassifier( max_depth = 8, criterion = 'gini')
rf.fit(x_train,y_train)

x_train_pred = rf.predict(x_train)
train_score = accuracy_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = rf.predict(x_val)
val_score = accuracy_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')

# scores are approximately near so no overfitting
```

```
the train score is =0.951271186440678
the valid score is =0.8559322033898306
```

In [42]:
```python
rf = RandomForestClassifier( max_depth = 9, criterion = 'gini')

rff = rf.fit(x_train,y_train)
y_pred = rff.predict(x_test)
```

In [43]:
```python
cnf_mat = confusion_matrix(y_test,y_pred)
sns.heatmap( cnf_mat , annot = True , cmap = 'Blues')
plt.title('confusion matrix of random forest')
```

Out[43]:
```
Text(0.5, 1.0, 'confusion matrix of random forest')
```



In [45]:

```
accuracy_random = accuracy_score(y_test, y_pred)
print(f'the accuracy of the random forest model is = {accuracy_random * 100} %')
recall = recall_score(y_test, y_pred)
print(f'the recall of the random forest model is = {recall * 100} %')
precision = precision_score(y_test, y_pred)
print(f'the precision of the random forest model is = {precision * 100} %')
f1 = f1_score(y_test, y_pred)
print(f'the f1_score of the random forest model is = {f1 * 100} %')
```

```
the accuracy of the random forest model is = 79.92125984251969 %
the recall of the random forest model is = 92.37288135593221 %
the precision of the random forest model is = 72.18543046357617 %
the f1_score of the random forest model is = 81.04089219330855 %
```

In [46]:

```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.91      0.69      0.79       136
           1       0.72      0.92      0.81       118

    accuracy                           0.80       254
   macro avg       0.82      0.81      0.80       254
weighted avg       0.82      0.80      0.80       254
```

SVM

In [48]:

```
# Use grid search to find best value
from sklearn.svm import SVC

svc = SVC(random_state = 42)

params_grid = {
    'C': [0.1,1,10],
    'kernel':['linear', 'rbf', 'poly'],
    'gamma':['scale', 'auto']
}

grid = GridSearchCV(
    svc,
    params_grid,
    cv = 5
```

```
)
grid.fit(x_train,y_train)

print(f'the best value is = {grid.best_params_}')
```

```
the best value is = {'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
```

In [49]:

```
## check overfitting

svc = SVC(C = 1, gamma = 'scale', kernel = 'linear')
svc.fit(x_train,y_train)

x_train_pred = svc.predict(x_train)
train_score = accuracy_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = svc.predict(x_val)
val_score = accuracy_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')

# scores are equal so no overfitting
```

```
the train score is =0.8453389830508474
the valid score is =0.8728813559322034
```

In [52]:

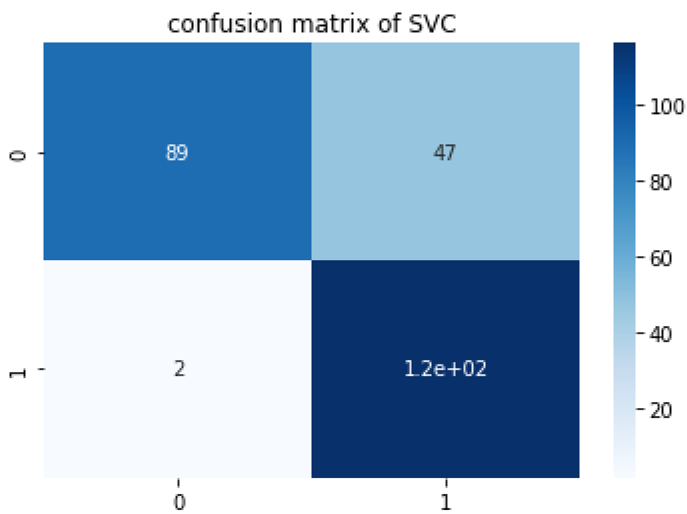
```
svc = SVC(C = 10, gamma = 'scale', kernel = 'linear')

svcc = svc.fit(x_train,y_train)
y_pred = svcc.predict(x_test)
```

In [53]:

```
cnf_mat = confusion_matrix(y_test,y_pred)
sns.heatmap( cnf_mat , annot = True , cmap = 'Blues')
plt.title('confusion matrix of SVC')
```

Out[53]:

```
Text(0.5, 1.0, 'confusion matrix of SVC')
```



In [54]:

```
accuracy_svc = accuracy_score(y_test, y_pred)
print(f'the accuracy of the SVC model is = {accuracy_svc * 100} %')
recall = recall_score(y_test, y_pred)
print(f'the recall of the SVC model is = {recall * 100} %')
precision = precision_score(y_test, y_pred)
print(f'the precision of the SVC model is = {precision * 100} %')
f1 = f1_score(y_test, y_pred)
print(f'the f1_score of the SVC model is = {f1 * 100} %')
```

```
the accuracy of the SVC model is = 80.70866141732283 %
the recall of the SVC model is = 98.30508474576271 %
the precision of the SVC model is = 71.16564417177914 %
the f1_score of the SVC model is = 82.56227758007118 %
```

In [55]:
```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.98      0.65      0.78       136
           1       0.71      0.98      0.83       118

    accuracy                           0.81       254
   macro avg       0.84      0.82      0.80       254
weighted avg       0.85      0.81      0.80       254
```

Logistic Regression

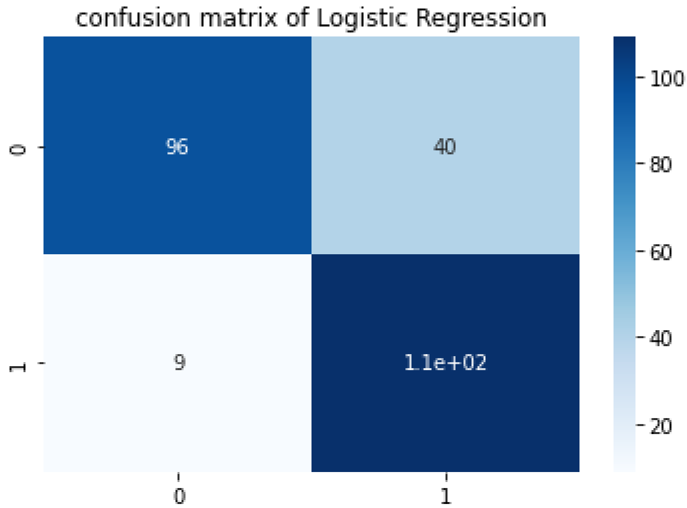
In [56]:
```
from sklearn.linear_model import LogisticRegression

## check overfitting

lr = LogisticRegression(random_state = 42)
lr.fit(x_train,y_train)

x_train_pred = lr.predict(x_train)
train_score = accuracy_score(y_train,x_train_pred)
print(f'the train score is ={train_score}')

x_val_pred = lr.predict(x_val)
val_score = accuracy_score(y_val,x_val_pred)
print(f'the valid score is ={val_score}')

# scores are equal so no overfitting
```

```
the train score is =0.864406779661017
the valid score is =0.864406779661017
```

In [57]:
```
lr = LogisticRegression(random_state = 42)

log = lr.fit(x_train,y_train)
y_pred = log.predict(x_test)
```

In [58]:
```
cnf_mat = confusion_matrix(y_test,y_pred)
sns.heatmap( cnf_mat , annot = True , cmap = 'Blues')
plt.title('confusion matrix of Logistic Regression')
```

Out[58]:
```
Text(0.5, 1.0, 'confusion matrix of Logistic Regression')
```



In [59]:
```
accuracy_log = accuracy_score(y_test, y_pred)
print(f'the accuracy of the logistic regression model is = {accuracy_log * 100} %')
recall = recall_score(y_test, y_pred)
print(f'the recall of the logistic regression model is = {recall * 100} %')
precision = precision_score(y_test, y_pred)
print(f'the precision of the logistic regression model is = {precision * 100} %')
f1 = f1_score(y_test, y_pred)
print(f'the f1_score of the logistic regression model is = {f1 * 100} %')
```

```
the accuracy of the logistic regression model is = 80.70866141732283 %
the recall of the logistic regression model is = 92.37288135593221 %
the precision of the logistic regression model is = 73.15436241610739 %
the f1_score of the logistic regression model is = 81.64794007490637 %
```

In [60]:
```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.91      0.71      0.80       136
           1       0.73      0.92      0.82       118

    accuracy                           0.81       254
   macro avg       0.82      0.81      0.81       254
weighted avg       0.83      0.81      0.81       254
```

Comparison of models evaluation (Accuracy)

In [61]:
```
models = ['KNN', 'Random Forest', 'SVM','Logistic Regression']
scores = [accuracy_knn, accuracy_random, accuracy_svc, accuracy_log]

plt.figure(figsize = (10,5))
sns.barplot(x = models, y = scores, data =df , palette = 'husl')
plt.title('accuracy scores of the models')
```

Out[61]:
```
Text(0.5, 1.0, 'accuracy scores of the models')
```

- We can conclude from the plot above that SVM and logistic regression classifier scored the highest accuracy.
- Random forest classifier classifier was much near to the accuracy scored by the SVM and logistic regression classifier.
- KNN Classifier had the lowest score among all of the classifiers.

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```