.....

AUTHOR - CHANDAN D.CHAUDHARI

GITHUB LINK - https://github.com/chandanc5525/TelecomChurn_PredictionModel

PROJECT NAME - TELECOM CHURN PREDICTION MODEL

.....

In [ ]:
```
# Import Python Libraries
import numpy as  np
import pandas as pd
# Importing Data Visualization Library
import seaborn as sns
import matplotlib.pyplot as plt
# Import Filter Warning Library
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# Importing Dataset using Pandas Function

URL = "https://raw.githubusercontent.com/chandanc5525/Dataset/main/Data/"

df = pd.read_csv(URL + 'telecom_churn.csv')

df.head(3)
```

Out[ ]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110

In [ ]:
```
df.shape
```

Out[ ]:
```
(3333, 20)
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
State                     0
Account length            0
Area code                 0
International plan        0
Voice mail plan           0
Number vmail messages     0
Total day minutes         0
Total day calls           0
Total day charge          0
Total eve minutes         0
Total eve calls           0
Total eve charge          0
Total night minutes       0
Total night calls         0
```

```
Total night charge        0
Total intl minutes        0
Total intl calls          0
Total intl charge         0
Customer service calls    0
Churn                     0
dtype: int64
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   State                   3333 non-null   object 
 1   Account length          3333 non-null   int64  
 2   Area code               3333 non-null   int64  
 3   International plan      3333 non-null   object 
 4   Voice mail plan         3333 non-null   object 
 5   Number vmail messages   3333 non-null   int64  
 6   Total day minutes       3333 non-null   float64
 7   Total day calls         3333 non-null   int64  
 8   Total day charge        3333 non-null   float64
 9   Total eve minutes       3333 non-null   float64
 10  Total eve calls         3333 non-null   int64  
 11  Total eve charge        3333 non-null   float64
 12  Total night minutes     3333 non-null   float64
 13  Total night calls       3333 non-null   int64  
 14  Total night charge      3333 non-null   float64
 15  Total intl minutes      3333 non-null   float64
 16  Total intl calls        3333 non-null   int64  
 17  Total intl charge       3333 non-null   float64
 18  Customer service calls  3333 non-null   int64  
 19  Churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 498.1+ KB
```

OBSERVATIONS

1. Total 3333 Rows and 20 Columns we have in our dataset.
2. No Null value present in the dataset.
3. Churn Column is Target Column so we need to convert dtype for this column to Number, We can do this conversion using map function.

In [ ]:
```
# Checking Columns
df.columns
```

Out[ ]:
```
Index(['State', 'Account length', 'Area code', 'International plan',
       'Voice mail plan', 'Number vmail messages', 'Total day minutes',
       'Total day calls', 'Total day charge', 'Total eve minutes',
       'Total eve calls', 'Total eve charge', 'Total night minutes',
       'Total night calls', 'Total night charge', 'Total intl minutes',
       'Total intl calls', 'Total intl charge', 'Customer service calls',
       'Churn'],
      dtype='object')
```

In [ ]:
```
feature = df[['International plan','Voice mail plan','Churn']]
for i in feature:
    print(i)
    print(feature[i].unique())
```

```
International plan
['No' 'Yes']
Voice mail plan
['Yes' 'No']
Churn
[False  True]
```

Important Note :-

1. International Plan if Yes than 1 else 0
2. Voice mail plan if Yes than 1 else 0
3. Churn if False than 0 else 1 i.e 0 value indicated people are loyal customers [People Do not Churn]
4. There are 51 States and area code is found to be only 3 i.e. 415, 408, 510.

In [ ]:
```
d = ({'Yes': 1, 'No': 0})


# Converting Categorical features into Numerical Features using map function

df['International plan'] = df['International plan'].map(d)
df['Voice mail plan'] = df['Voice mail plan'].map(d)
df["Churn"] = df["Churn"].astype("int64")     # for converting boolean datat


df.head()
```

Out[ ]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls
0	KS	128	415	0	1	25	265.1	110	45.07	197.4	99
1	OH	107	415	0	1	26	161.6	123	27.47	195.5	103
2	NJ	137	415	0	0	0	243.4	114	41.38	121.2	110
3	OH	84	408	1	0	0	299.4	71	50.90	61.9	88
4	OK	75	415	1	0	0	166.7	113	28.34	148.3	122

In [ ]:
```
# Checking Descriptive Stats

df.describe()
```

Out[ ]:

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total c ca
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0000
mean	101.064806	437.182418	0.096910	0.276628	8.099010	179.775098	100.4356
std	39.822106	42.371290	0.295879	0.447398	13.688365	54.467389	20.0690
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	74.000000	408.000000	0.000000	0.000000	0.000000	143.700000	87.0000
50%	101.000000	415.000000	0.000000	0.000000	0.000000	179.400000	101.0000
75%	127.000000	510.000000	0.000000	1.000000	20.000000	216.400000	114.0000
max	243.000000	510.000000	1.000000	1.000000	51.000000	350.800000	165.0000

In [ ]:
```
# Churn Information
print('--------------------------------')
print(df['Churn'].value_counts())
print(df['Churn'].value_counts(normalize=True))      # Shows the value in ter
print('--------------------------------')
```

```
--------------------------------
0    2850
1     483
Name: Churn, dtype: int64
0    0.855086
1    0.144914
Name: Churn, dtype: float64
--------------------------------
```

In [ ]:
```
plt.figure(figsize = (30,10))

plt.show()
```

```
<Figure size 3000x1000 with 0 Axes>
```

In [ ]:
```
df[df["Churn"] == 1]["Total day minutes"].mean()
```

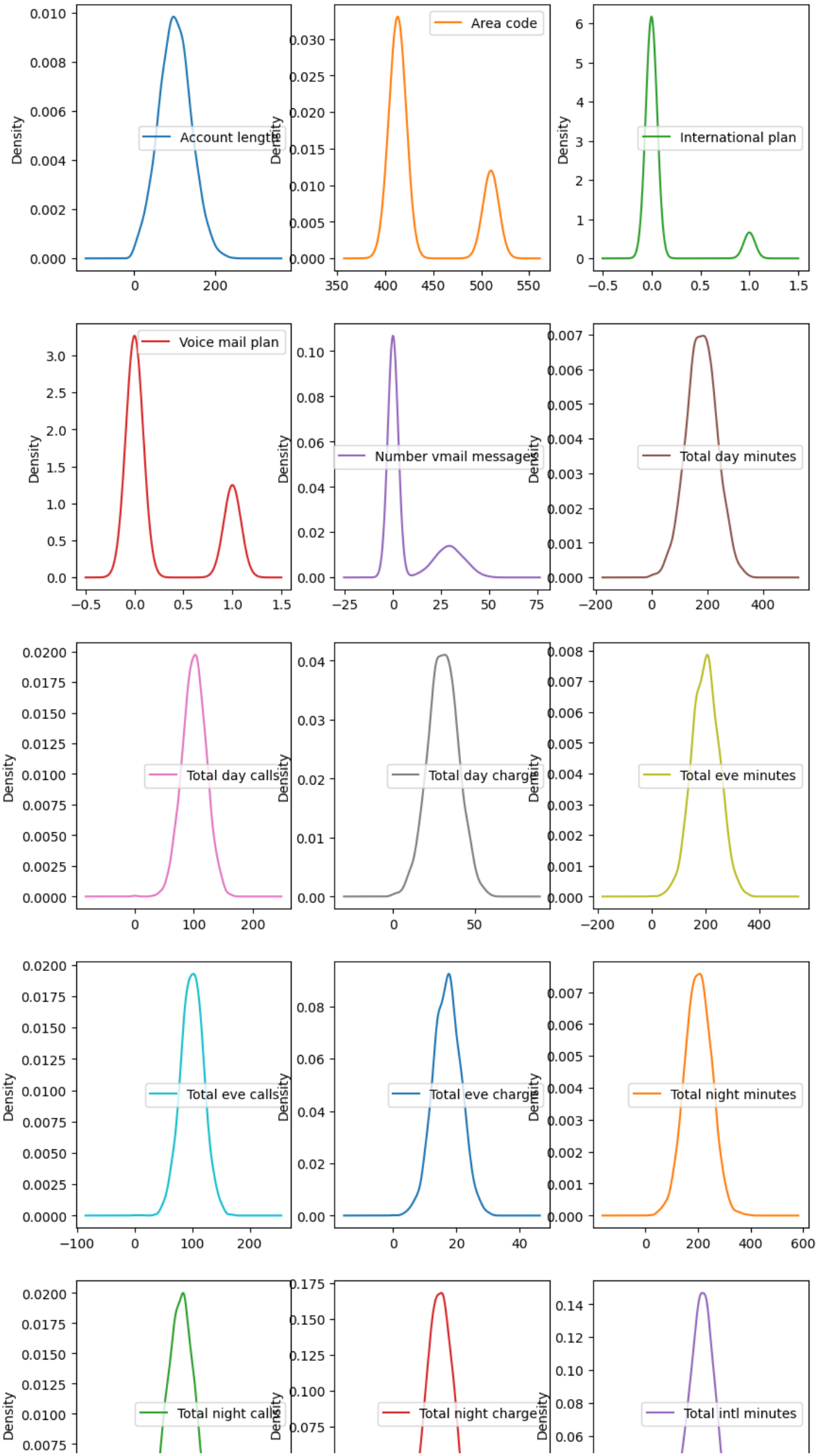
Out[ ]:
```
206.91407867494823
```

In [ ]:
```
df[(df["Churn"] == 0) & (df["International plan"] == 0)]["Total intl minutes
```

Out[ ]:
```
18.9
```

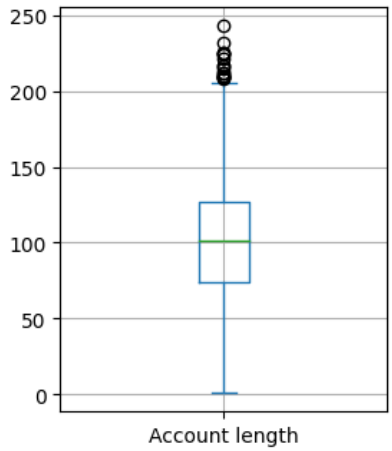
In [ ]:
```
plt.figure(figsize=(15,20))
df.plot(kind='kde', subplots=True, layout=(7,3), sharex=False,sharey=False,f
```

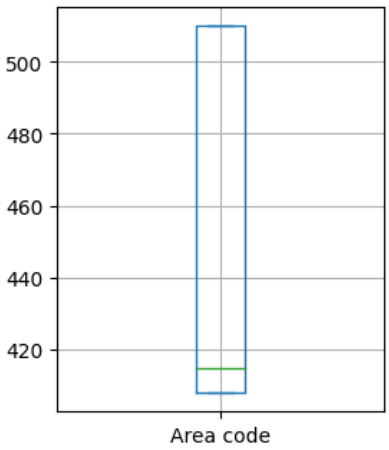
```
<Figure size 1500x2000 with 0 Axes>
```

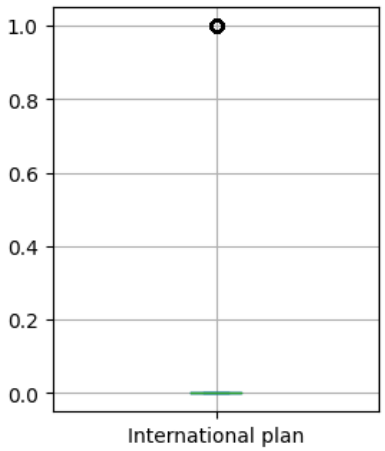

Account length
Area code
International plan
Voice mail plan
Number vmail messages
Total day minutes
Total day calls
Total day charge
Total eve minutes
Total eve calls
Total eve charge
Total night minutes
Total night calls
Total night charge
Total intl minutes
Density

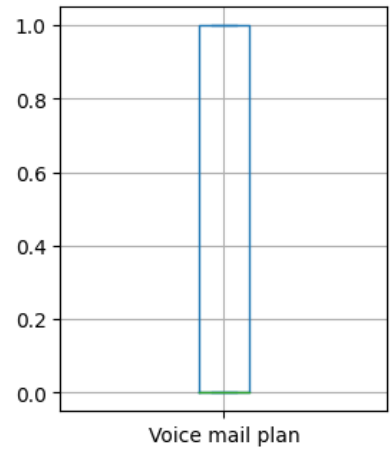
In [ ]:
```
# Box Plot Before Treating an Outliers
plt.figure(figsize=(15,20))
df.plot(kind='box', subplots=True, layout=(7,3), sharex=False,sharey=False,f
plt.show()
```

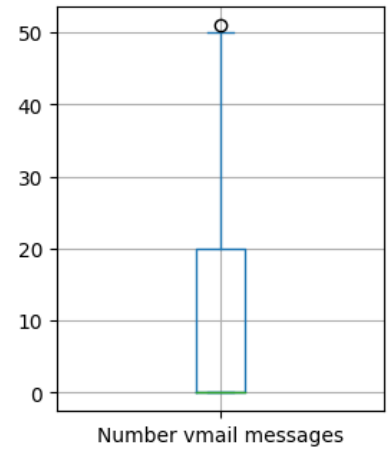
```
<Figure size 1500x2000 with 0 Axes>
```

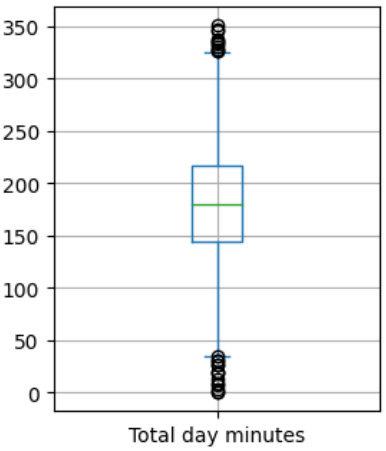

250
200
150
100
50
0
Account length

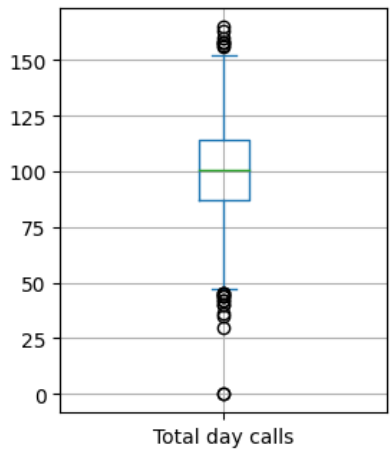

500
480
460
440
420
Area code

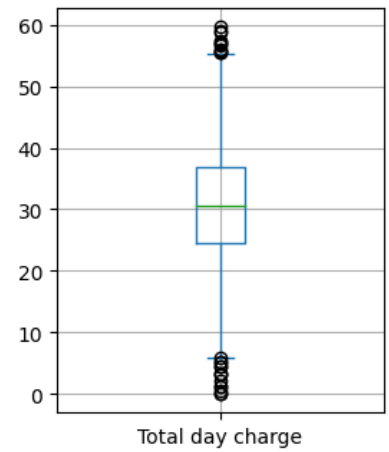

1.0
0.8
0.6
0.4
0.2
0.0
International plan

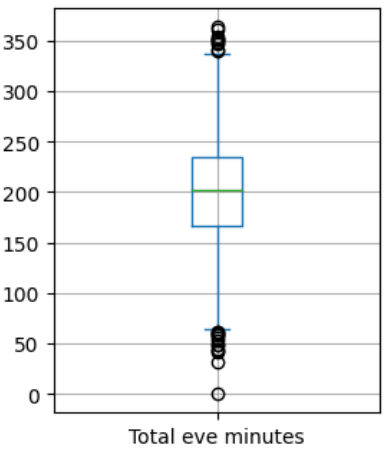

1.0
0.8
0.6
0.4
0.2
0.0
Voice mail plan

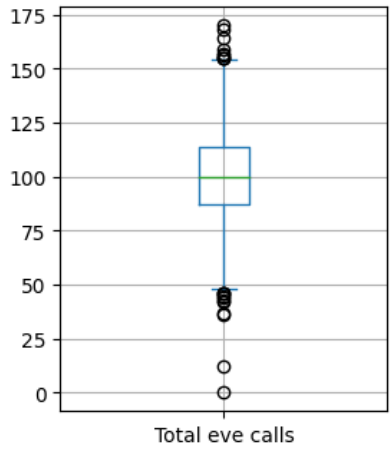

50
40
30
20
10
0
Number vmail messages

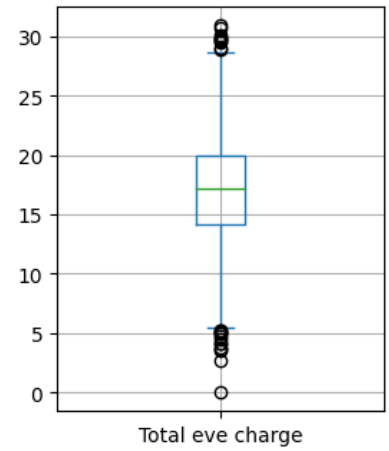

350
300
250
200
150
100
50
0
Total day minutes

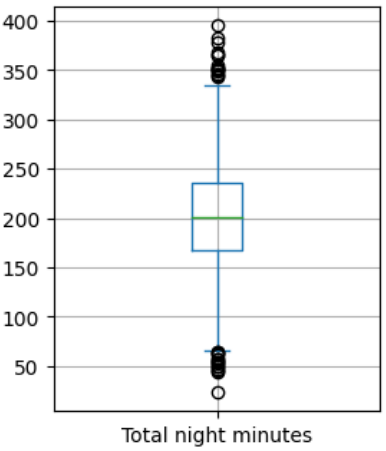

150
125
100
75
50
25
0
Total day calls

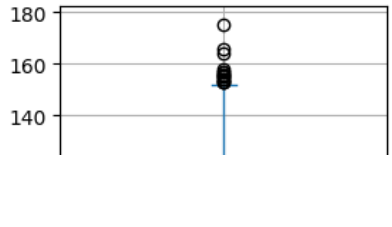

60
50
40
30
20
10
0
Total day charge

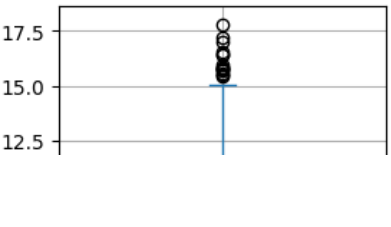

350
300
250
200
150
100
50
0
Total eve minutes

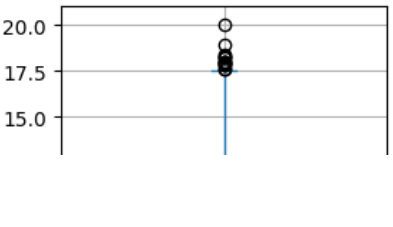

175
150
125
100
75
50
25
0
Total eve calls


30
25
20
15
10
5
0
Total eve charge


400
350
300
250
200
150
100
50
Total night minutes


180
160
140


17.5
15.0
12.5


20.0
17.5
15.0

In [ ]:
```
# Creating a new variable without Target Column
features = df.drop('Churn',axis=1)
features
```

Out[ ]:

	State	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	T c
0	KS	128	415	0	1	25	265.1	110	45.07	197.4	
1	OH	107	415	0	1	26	161.6	123	27.47	195.5	
2	NJ	137	415	0	0	0	243.4	114	41.38	121.2	
3	OH	84	408	1	0	0	299.4	71	50.90	61.9	
4	OK	75	415	1	0	0	166.7	113	28.34	148.3	
...	...	...	...	...	...	...	...	...	...	...	
3328	AZ	192	415	0	1	36	156.2	77	26.55	215.5	
3329	WV	68	415	0	0	0	231.1	57	39.29	153.4	
3330	RI	28	510	0	0	0	180.8	109	30.74	288.8	
3331	CT	184	510	1	0	0	213.8	105	36.35	159.6	

3332	TN	74	415	0	1	25	234.4	113	39.85	265.9	

3333 rows × 19 columns

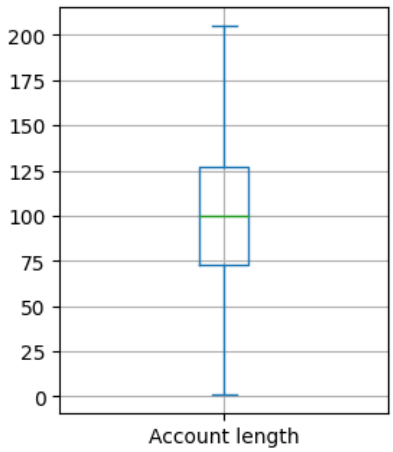
In [ ]:
```
# Identify and handle outliers
# For example, we can use the Interquartile Range (IQR) method to remove out
Q1 = features.quantile(0.25)
Q3 =  features.quantile(0.75)
IQR = Q3 - Q1
data= features[~((features < (Q1 - 1.5 * IQR)) | (features > (Q3 + 1.5 * IQR
```

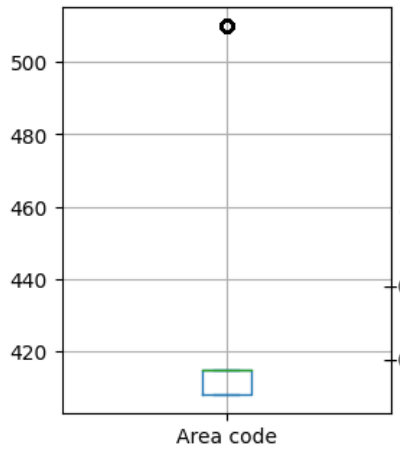
In [ ]:
```
data.shape
```

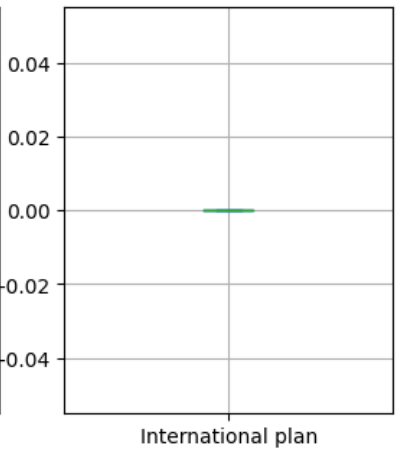
Out[ ]:
```
(2536, 19)
```

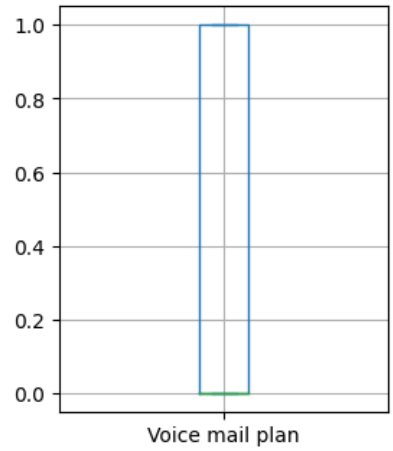
In [ ]:
```
# Box plot for the dataset after outlier Removal
plt.figure(figsize=(15,20))
data.plot(kind='box', subplots=True, layout=(7,3), sharex=False,sharey=False
plt.show()
```

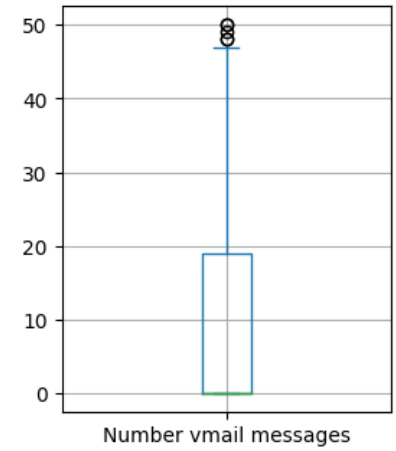
```
<Figure size 1500x2000 with 0 Axes>
```

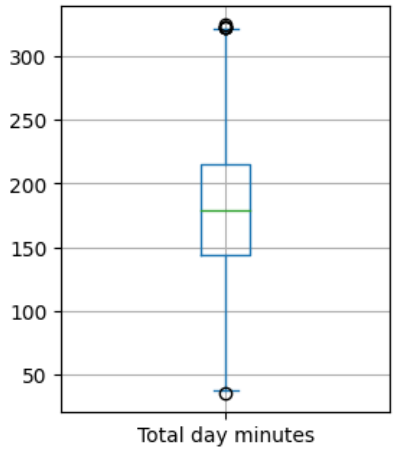

200
175
150
125
100
75
50
25
0
Account length

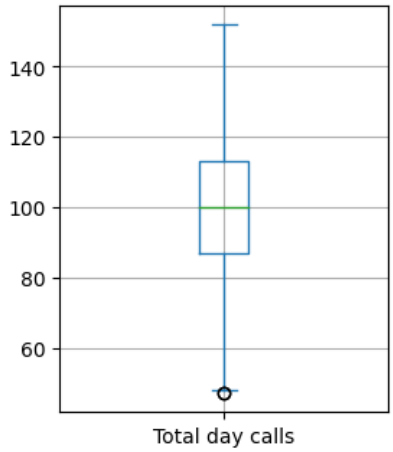

500
480
460
440
420
Area code

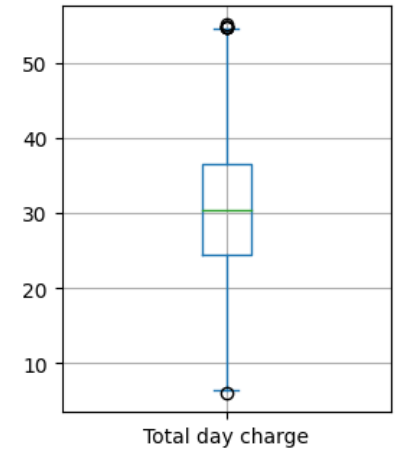

0.04
0.02
0.00
−0.02
−0.04
International plan

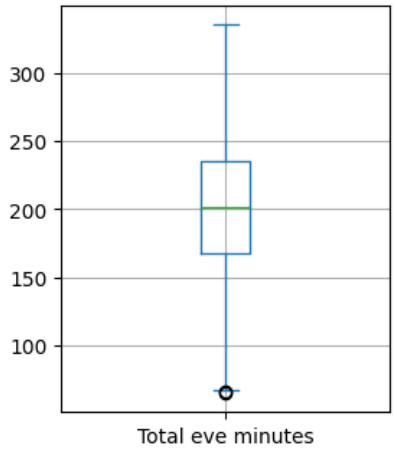

1.0
0.8
0.6
0.4
0.2
0.0
Voice mail plan

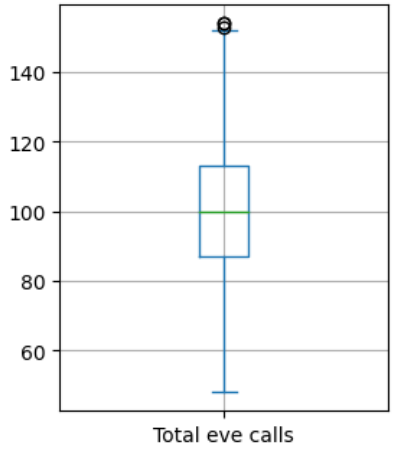

50
40
30
20
10
0
Number vmail messages

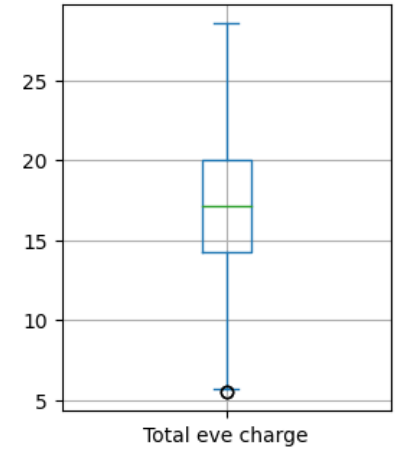

300
250
200
150
100
50
Total day minutes

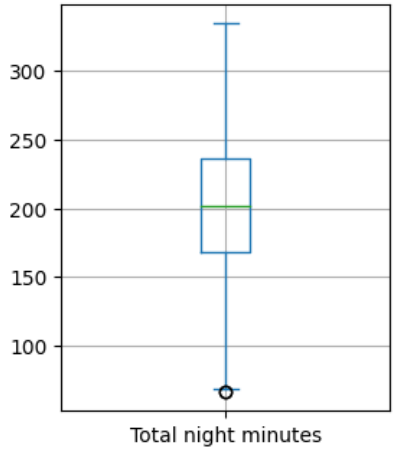

140
120
100
80
60
Total day calls

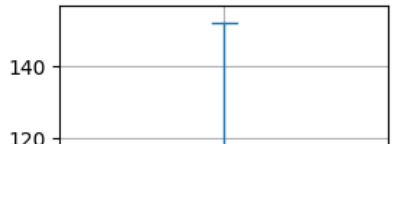

50
40
30
20
10
Total day charge

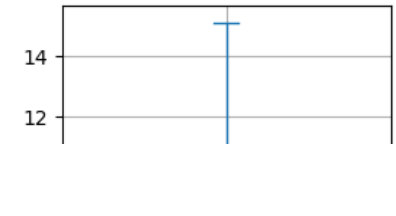

300
250
200
150
100
Total eve minutes

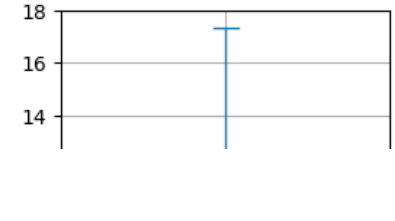

140
120
100
80
60
Total eve calls


25
20
15
10
5
Total eve charge

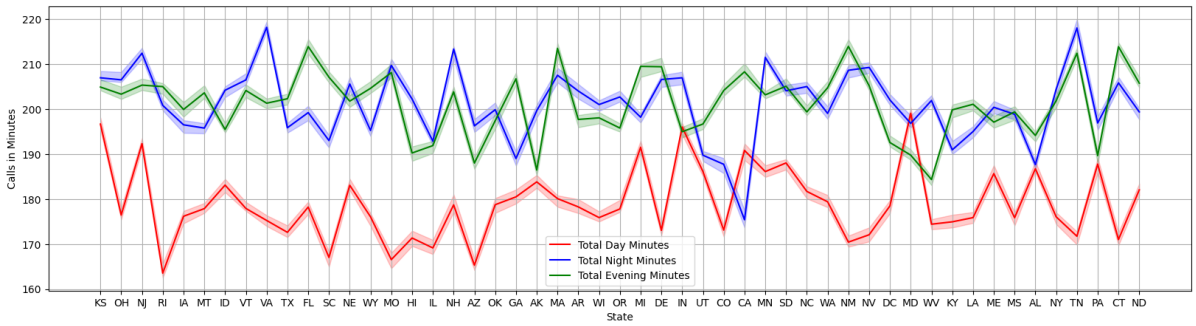

300
250
200
150
100
Total night minutes


140
120

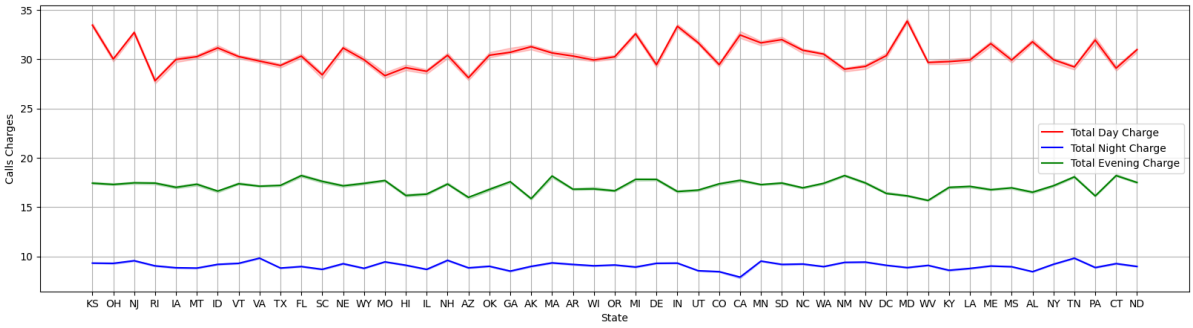

14
12


18
16
14

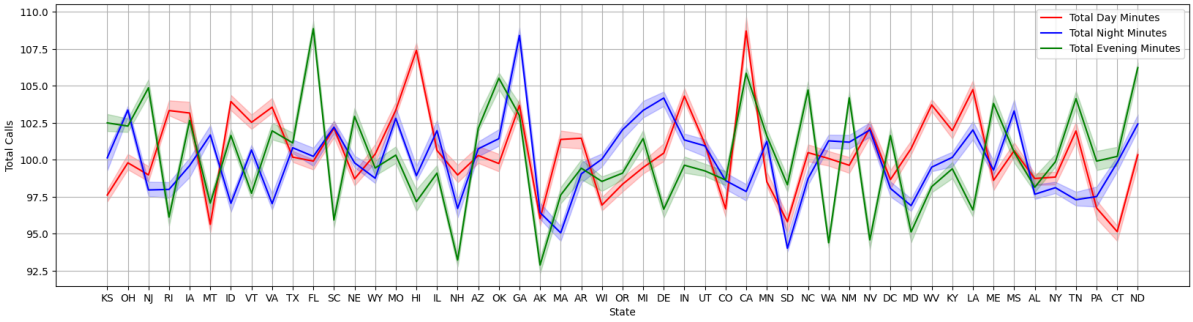
In [ ]:
```
#  Visualization for Calls in Minutes vs State

plt.figure(figsize=(20,5))
plt.grid()
sns.lineplot(x=data['State'],y=data['Total day minutes'],label='Total Day Mi
sns.lineplot(x=data['State'],y=data['Total night minutes'],label='Total Nigh
sns.lineplot(x=data['State'],y=data['Total eve minutes'],label='Total Evenin
plt.ylabel('Calls in Minutes')
plt.xlabel('State')
plt.show()
```



In [ ]:
```
# Visualization for Call Charges vs State

plt.figure(figsize=(20,5))
plt.grid()
sns.lineplot(x=data['State'],y=data['Total day charge'],label='Total Day Cha
sns.lineplot(x=data['State'],y=data['Total night charge'],label='Total Night
sns.lineplot(x=data['State'],y=data['Total eve charge'],label='Total Evening
plt.ylabel('Calls Charges')
plt.xlabel('State')
plt.show()
```

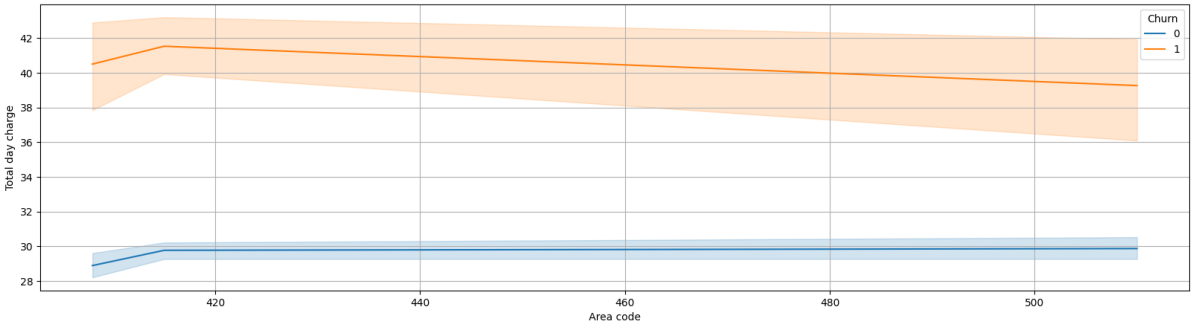


In [ ]:
```
# Visualization for Total Number of Calls vs State

plt.figure(figsize=(20,5))
plt.grid()
sns.lineplot(x=data['State'],y=data['Total day calls'],label='Total Day Minu
sns.lineplot(x=data['State'],y=data['Total night calls'],label='Total Night
sns.lineplot(x=data['State'],y=data['Total eve calls'],label='Total Evening
plt.ylabel('Total Calls')
plt.xlabel('State')
plt.show()
```



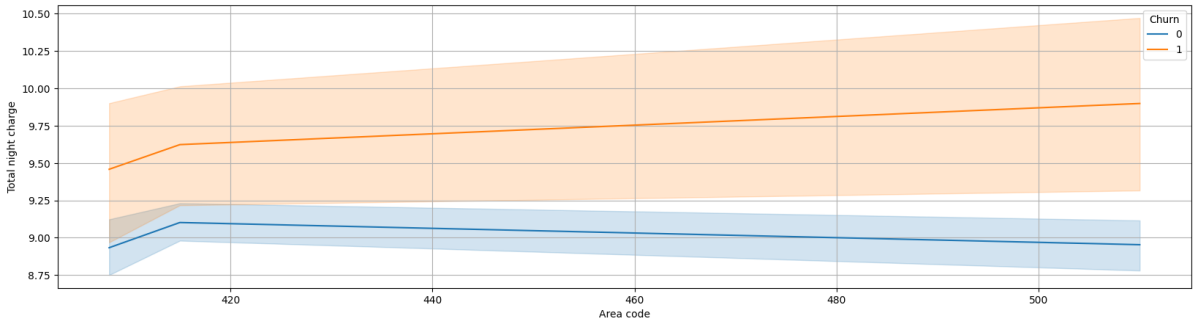
In [ ]:
```
# Visualization for Total Day Charge vs Area Code

plt.figure(figsize=(20,5))
plt.grid()
sns.lineplot(x=data['Area code'],y=data['Total day charge'],hue= df['Churn']
```

Out[ ]:
```
<AxesSubplot: xlabel='Area code', ylabel='Total day charge'>
```



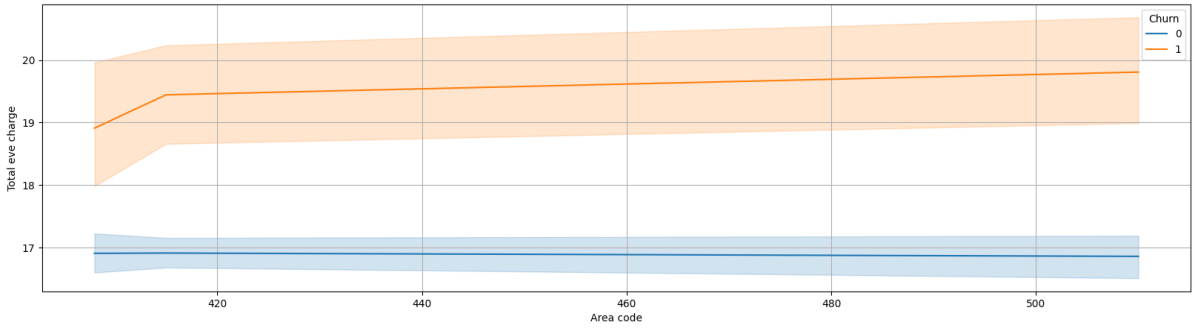
In [ ]:
```
# Visualization for Total Night Charge vs Area Code

plt.figure(figsize=(20,5))
plt.grid()
sns.lineplot(x=data['Area code'],y=data['Total night charge'],hue= df['Churn
```

Out[ ]:
```
<AxesSubplot: xlabel='Area code', ylabel='Total night charge'>
```



In [ ]:
```
# Visualization for Total Evening Charge vs Area Code

plt.figure(figsize=(20,5))
plt.grid()
sns.lineplot(x=data['Area code'],y=data['Total eve charge'],hue= df['Churn']
```

Out[ ]:
```
<AxesSubplot: xlabel='Area code', ylabel='Total eve charge'>
```



In [ ]:
```
# Final Dataset

dataset = data.join(df['Churn'], how = 'left')
dataset.drop('State',axis=1,inplace=True)
```

In [ ]:
```
dataset.head()
```

Out[ ]:

	Account length	Area code	International plan	Voice mail plan	Number vmail messages	Total day minutes	Total day calls	Total day charge	Total eve minutes	Total eve calls	To e char
0	128	415	0	1	25	265.1	110	45.07	197.4	99	16
1	107	415	0	1	26	161.6	123	27.47	195.5	103	16
2	137	415	0	0	0	243.4	114	41.38	121.2	110	10
11	74	415	0	0	0	187.7	127	31.91	163.4	148	13
12	168	408	0	0	0	128.8	96	21.90	104.9	71	8

In [ ]:
```
# Importing Machine Learning Libraries

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score,precision_score
```

In [ ]:
```
# Splitting the Dataset into Independent and Dependent Column

X =  dataset.iloc[:,:-1]
y = dataset.iloc[:,-1:]
```

In [ ]:
```
# Using MinMax Scaler Technique

X = MinMaxScaler().fit_transform(X)
```

In [ ]:
```
# Split the Dataset as train and test sets

X_train,X_test,y_train,y_test = train_test_split(X,y,random_state=42,test_si
```

In [ ]:
```
# Model Evaluation

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score,RandomizedSearchCV
from sklearn.metrics import accuracy_score,mean_squared_error,mean_absolute_

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.3)
```

In [ ]:
```
# Import Sklearn Models

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

models = {'LogisticRegression':LogisticRegression(),
          'RandomForestClassifier':RandomForestClassifier(),
          'KNNClassifier':KNeighborsClassifier()}

def evaluate(models,X_train,X_test,y_train,y_test):
    np.random.seed(42)
    # Creating One Dictionary to Save Model Score
    model_score = {}
    for name,model in models.items():
        model.fit(X_train, y_train)
        model_score[name] = model.score(X_test,y_test)
    return model_score
```

In [ ]:
```
model_score = evaluate(models = models ,X_train = X_train,X_test = X_test,y_
model_score
```

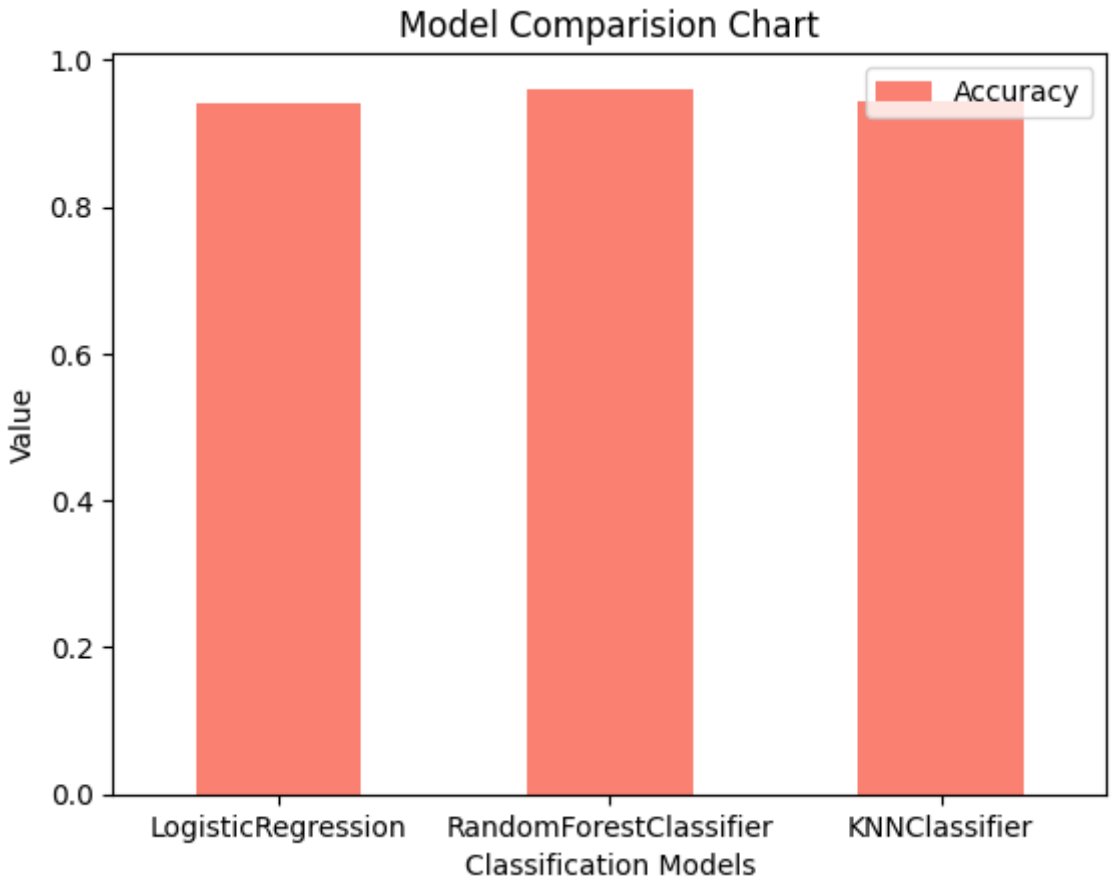
Out[ ]:
```
{'LogisticRegression': 0.9395532194480947,
 'RandomForestClassifier': 0.9605781865965834,
 'KNNClassifier': 0.9434954007884363}
```

In [ ]:
```
# Model Comparison

model_comparison = pd.DataFrame(model_score,index = ['Accuracy'])
model_comparison.sort_values(by = 'Accuracy',axis = 1,ascending= False)
```

Out[ ]:

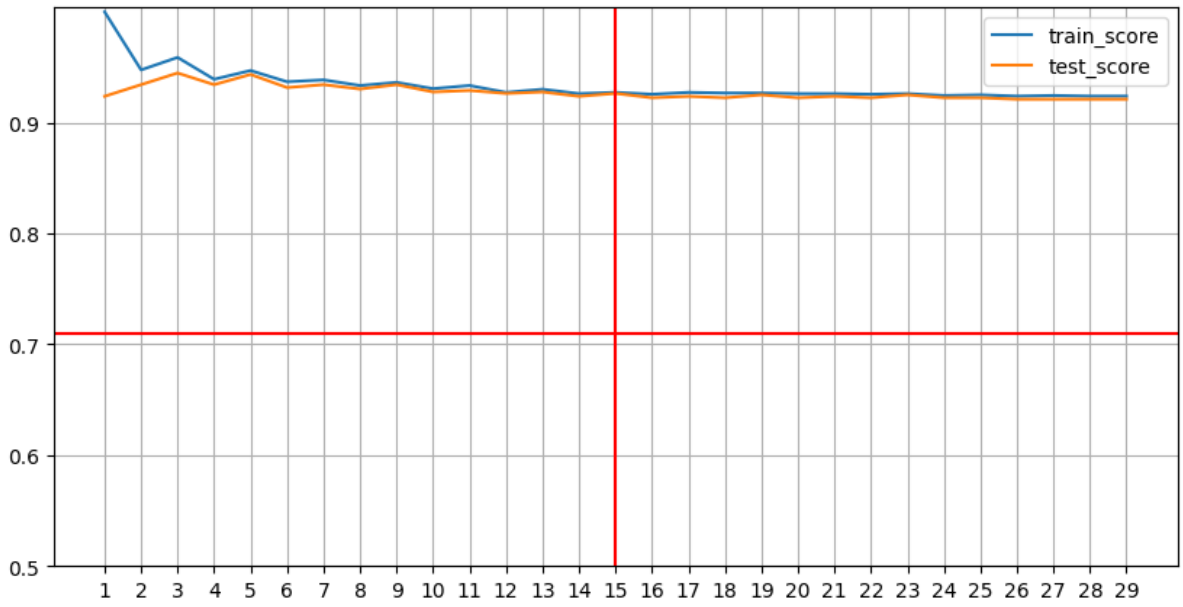
	RandomForestClassifier	KNNClassifier	LogisticRegression
Accuracy	0.960578	0.943495	0.939553

In [ ]:
```
# Model Comparison Graphical Representation

model_comparison.T.plot(kind = 'bar',color = 'salmon')
plt.title('Model Comparision Chart')
plt.xlabel('Classification Models')
plt.ylabel('Value')
plt.xticks(rotation = 0)
plt.show()
```



In [ ]:

```
# Tunning KNN Model

train_score = []
test_score = []

# Create a list of different values of n_neighbours
neighbors = range(1,30)

# Setting KNN
Knn = KNeighborsClassifier()

# Loop Calcualtions
for i in neighbors:
    Knn.set_params(n_neighbors = i)
    Knn.fit(X_train,y_train)

    train_score.append(Knn.score(X_train,y_train))
    test_score.append(Knn.score(X_test,y_test))
plt.subplots(figsize = (10,5))
plt.plot(neighbors,train_score)
plt.plot(neighbors,test_score)
plt.xticks(np.arange(1,30,1))
plt.yticks(np.arange(0.5,1,0.1))
plt.axhline(0.71,c = 'r')
plt.axvline(15,c = 'r')
plt.legend(['train_score','test_score'])
plt.grid()
plt.show()
```



In [ ]:
```
rf = RandomForestClassifier()
rf.get_params()   # Checking Various Parameters for RandomForestClassifier
```

Out[ ]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [ ]:
```
rf_grid = { 'n_estimators': np.arange(10,1000,50),
            'max_depth': [None,3,5,10],
            'min_samples_leaf': np.arange(2,20,2),
            'min_samples_split': np.arange(1,20,2)
          }

np.random.seed(42)

randomforest = RandomizedSearchCV(RandomForestClassifier(),param_distributio
randomforest.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

Out[ ]:
```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [ ]:
```
randomforest.best_params_   # Best Parameter for RandomForestClassifier Mode
```

Out[ ]:
```
{'n_estimators': 910,
 'min_samples_split': 17,
 'min_samples_leaf': 2,
 'max_depth': 10}
```

In [ ]:
```
randomforest.score(X_test,y_test)
```

Out[ ]:
```
0.9632063074901446
```

In [ ]:
```
print(f'The Model Score using RandomForestClassifier is {randomforest.score(
```

```
The Model Score using RandomForestClassifier is 96.32 %
```

In [ ]:
```
# Classification Report

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

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       698
           1       0.97      0.57      0.72        63

    accuracy                           0.96       761
   macro avg       0.97      0.78      0.85       761
weighted avg       0.96      0.96      0.96       761
```

In [ ]:
```
cv_acc = cross_val_score(randomforest,X,y,cv = 5,scoring = 'accuracy')
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

In [ ]:
```
print(f'CV Accuracy Score : {np.mean(cv_acc)*100:.2f} %')
```

```
CV Accuracy Score : 95.98 %
```

In [ ]:
```
cv_prec = cross_val_score(randomforest,X,y,cv = 5,scoring = 'precision')
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

In [ ]:
```
print(f'CV Precision Score : {np.mean(cv_prec)*100:.2f} %')
```

```
CV Precision Score : 95.17 %
```

In [ ]:
```
cv_recall = cross_val_score(randomforest,X,y,cv = 5,scoring = 'recall')
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

In [ ]:
```
print(f'CV Recall Score : {np.mean(cv_recall)*100:.2f} %')
```

```
CV Recall Score : 54.23 %
```

In [ ]:
```
cv_f1 = cross_val_score(randomforest,X,y,cv = 5,scoring = 'f1')
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

In [ ]:
```
print(f'CV F1 Score : {np.mean(cv_f1)*100:.2f} %')
```

```
CV F1 Score : 66.95 %
```

In [ ]:
```
CrossvalidationData = pd.DataFrame({'Accuracy':np.mean(cv_acc),
                                    'Precision':np.mean(cv_prec),
                                    'Recall':np.mean(cv_recall),
                                    'F1 Score':np.mean(cv_f1)
                                     },index = [0])
CrossvalidationData.T.plot(kind='bar',color = 'lightblue')
plt.xticks(rotation=0)
plt.show()
```