In [2]:
```
#important packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
sns.set_style('darkgrid')
```

In [3]:
```
# machine learning models
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifie
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
```

In [66]:
```
#Load data
df = pd.read_csv('./data.csv')
df.columns
```

Out[66]:
```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

In [67]:
```
df.shape
```

Out[67]:
```
(7043, 21)
```

In [68]:
```
df.head(5)
```

Out[68]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines |
|---|---|---|---|---|---|---|---|---|
| **0** | 7590-VHVEG | Female | 0 | Yes | No | 1 | No | No phone service |
| **1** | 5575-GNVDE | Male | 0 | No | No | 34 | Yes | No |
| **2** | 3668-QPYBK | Male | 0 | No | No | 2 | Yes | No |
| **3** | 7795-CFOCW | Male | 0 | No | No | 45 | No | No phone service |
| **4** | 9237-HQITU | Female | 0 | No | No | 2 | Yes | No |

5 rows × 21 columns

In [69]:
```
df.isna().sum()
```

Out[69]:
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

In [70]:
```
df.info()
```

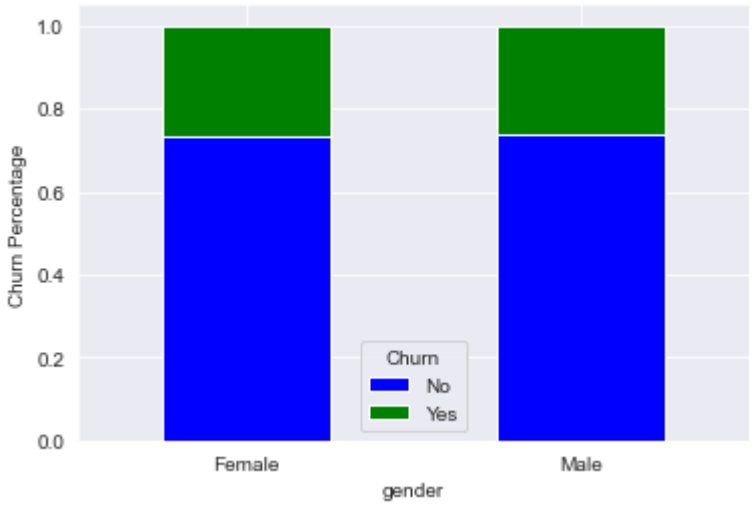
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

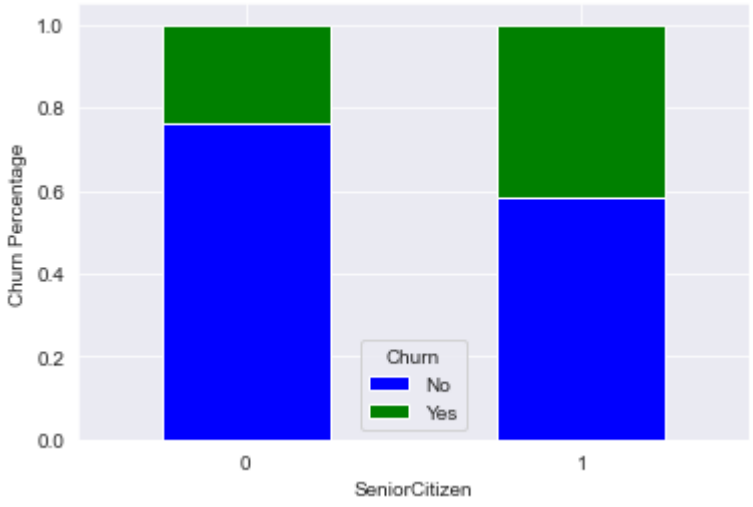
# EDA

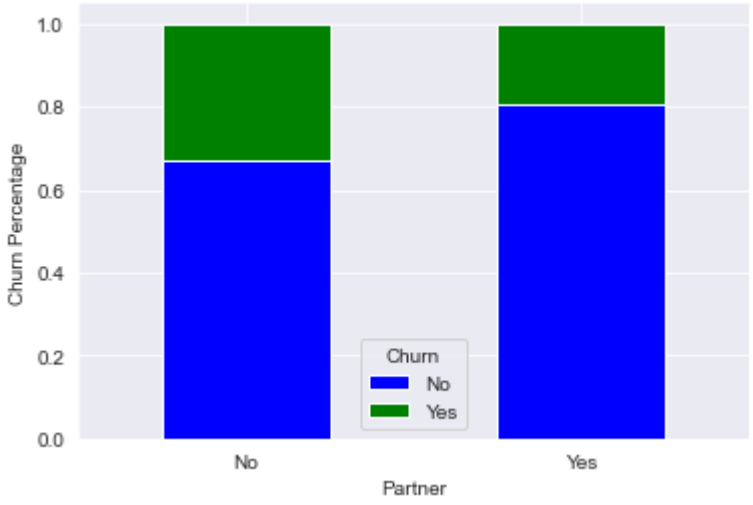
In [71]:
```
def stacked_plot(df, group, target):
    """
    Function to generate a stacked plots between two variables
    """
    fig, ax = plt.subplots(figsize = (6,4))
    temp_df = (df.groupby([group, target]).size()/df.groupby(group)[target].co
    temp_df.plot(kind='bar', stacked=True, ax = ax, color = ["blue", "green"])
    ax.xaxis.set_tick_params(rotation=0)
    ax.set_xlabel(group)
    ax.set_ylabel('Churn Percentage')
```

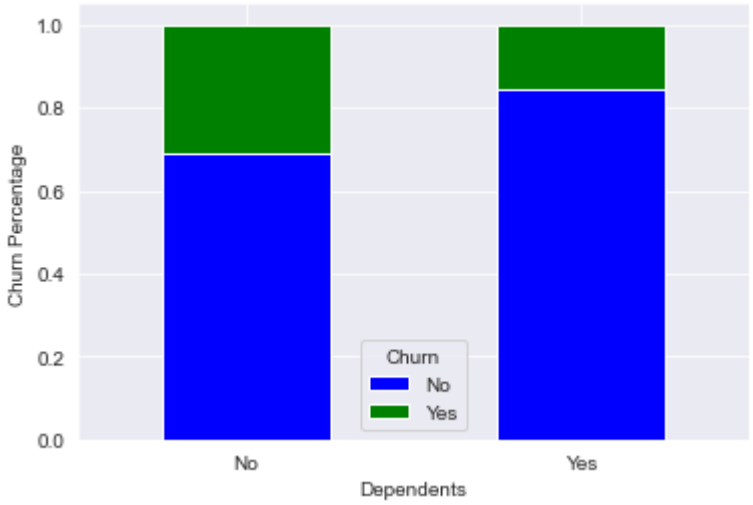
**For gender, SeniorCitizen, Partner, Dependents**

In [72]:
```
stacked_plot(df, "gender", "Churn")
stacked_plot(df, "SeniorCitizen", "Churn")
stacked_plot(df, "Partner", "Churn")
stacked_plot(df, "Dependents", "Churn")
```



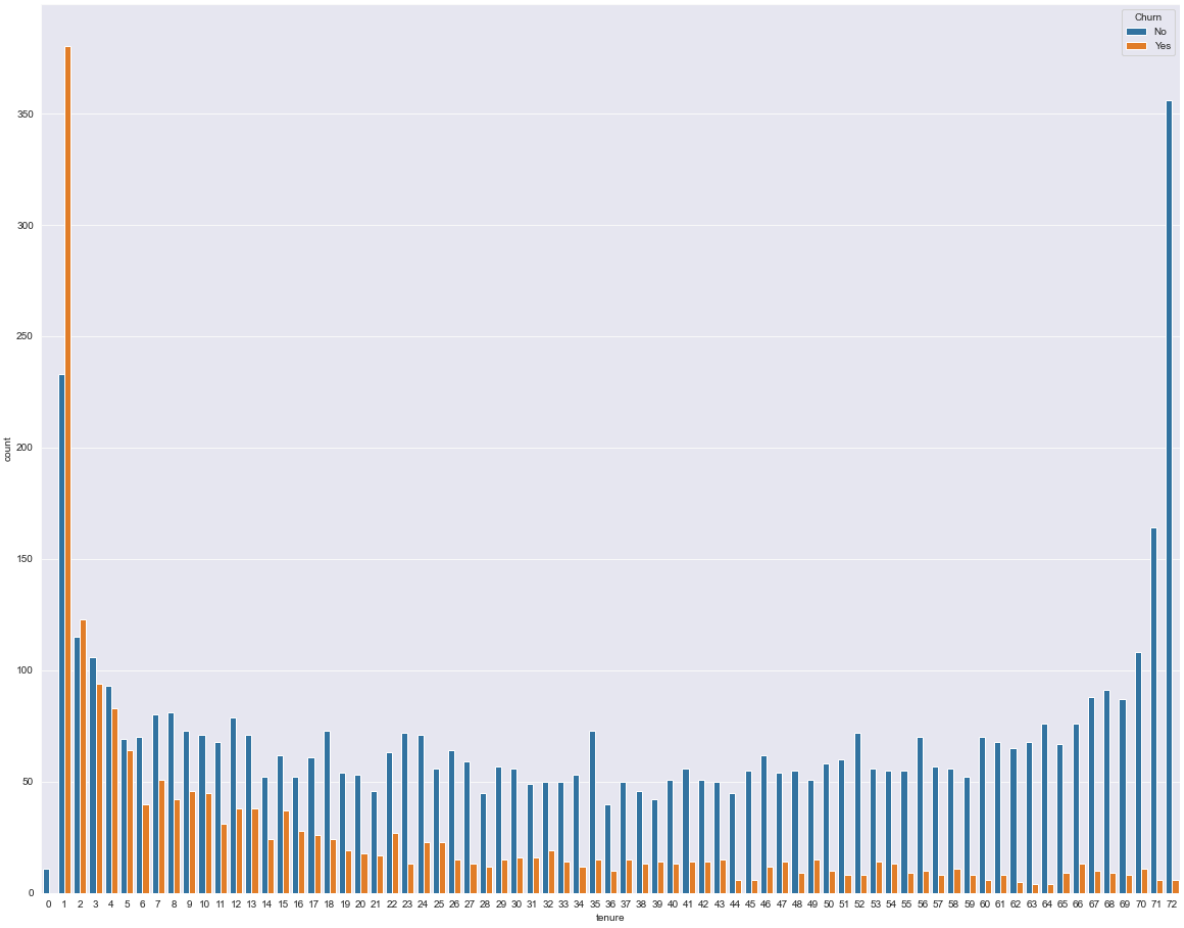





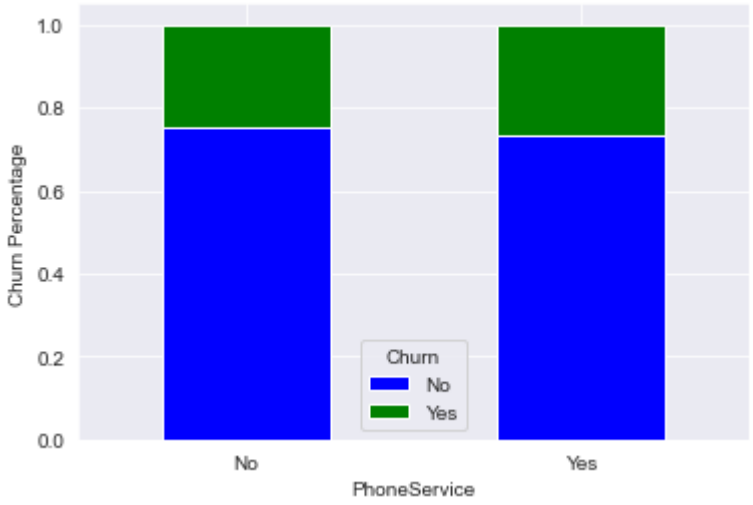

**For Tenure**

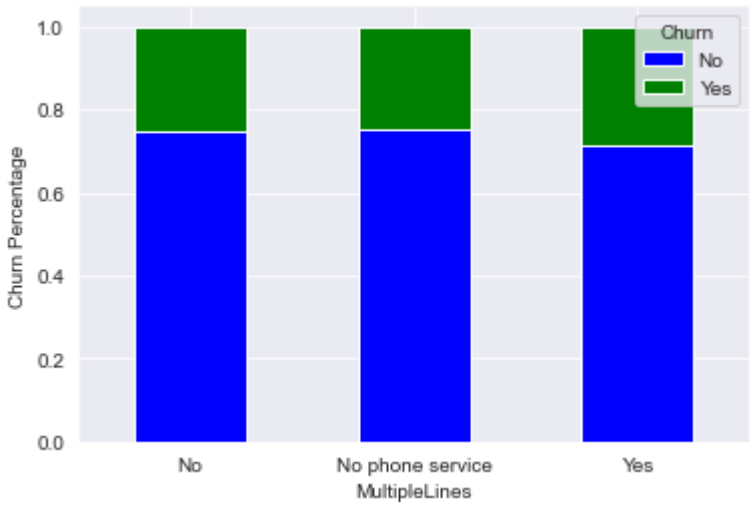
In [75]:
```
plt.figure(figsize=(20,16))
sns.countplot(x="tenure", hue="Churn", data=df)
plt.show()
```




**Phone service and multipleLines**

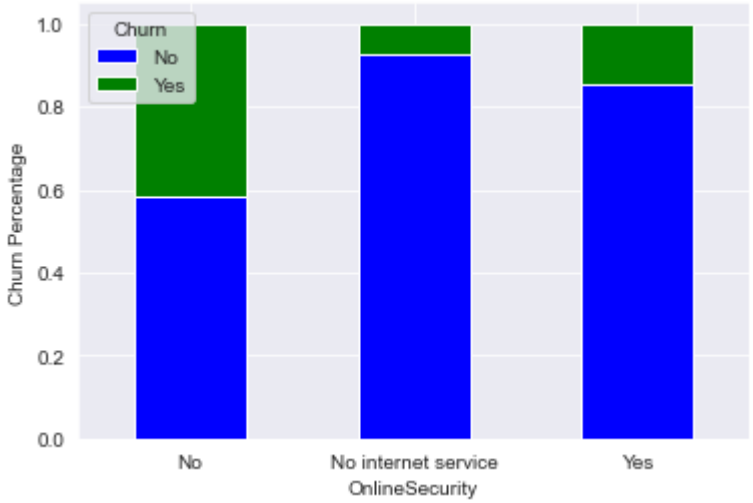
In [74]:
```
stacked_plot(df, "PhoneService", "Churn")
stacked_plot(df, "MultipleLines", "Churn")
```

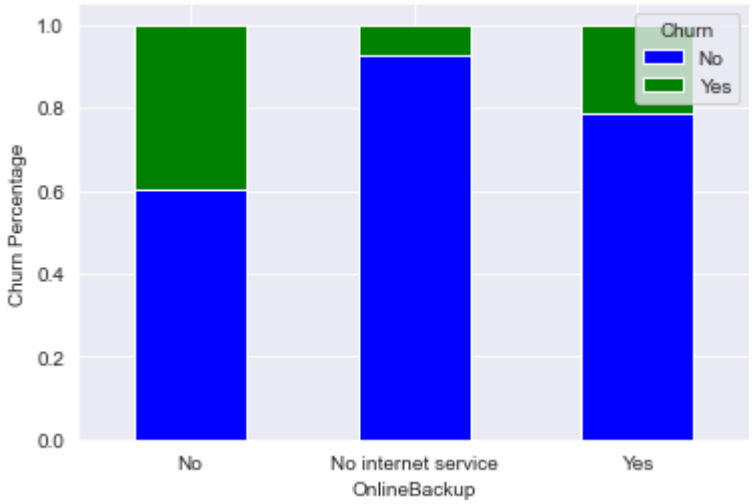


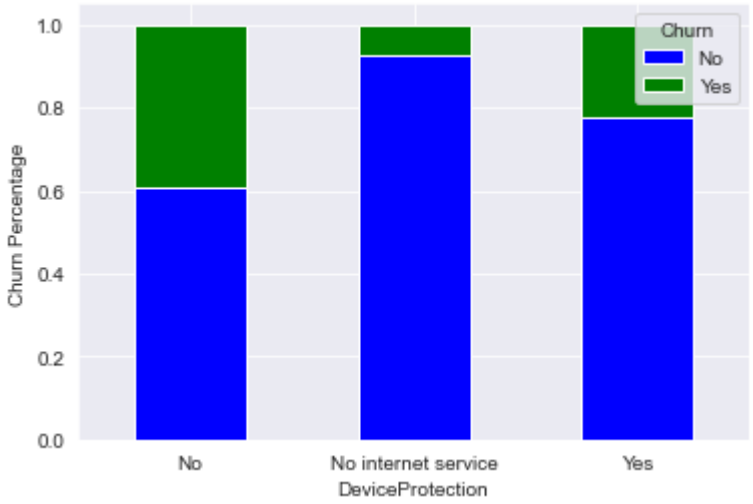



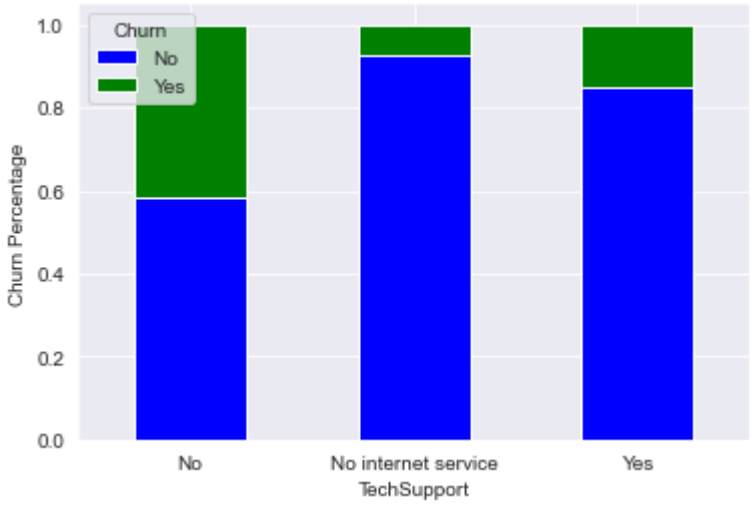

**OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies**

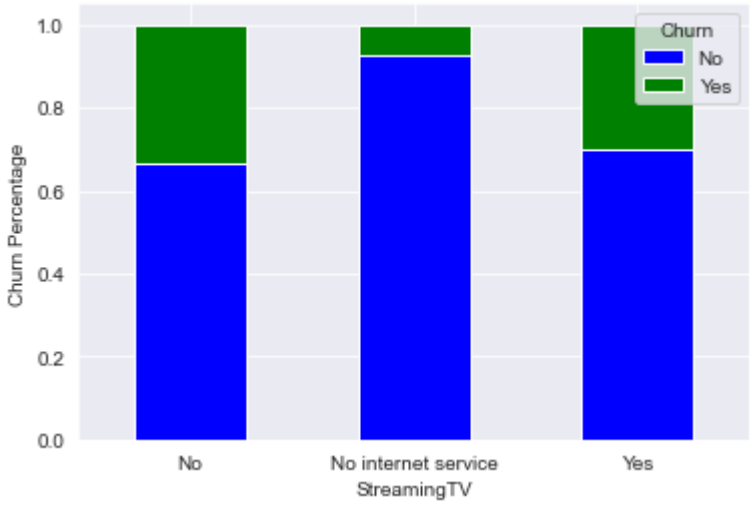
In [76]:
```
stacked_plot(df, "OnlineSecurity", "Churn")
stacked_plot(df, "OnlineBackup", "Churn")
stacked_plot(df, "DeviceProtection", "Churn")
stacked_plot(df, "TechSupport", "Churn")
stacked_plot(df, "StreamingTV", "Churn")
stacked_plot(df, "StreamingMovies", "Churn")
```

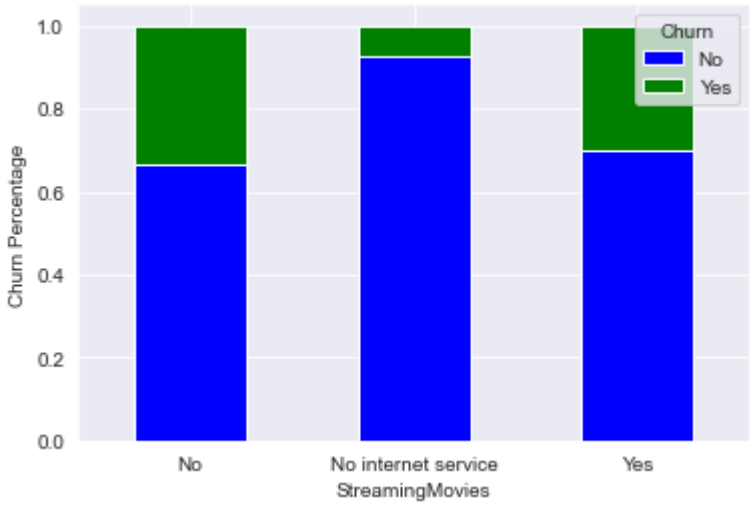


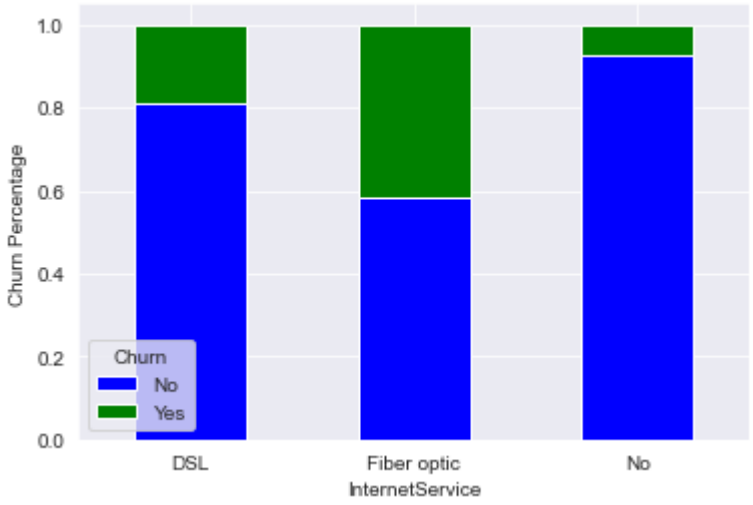


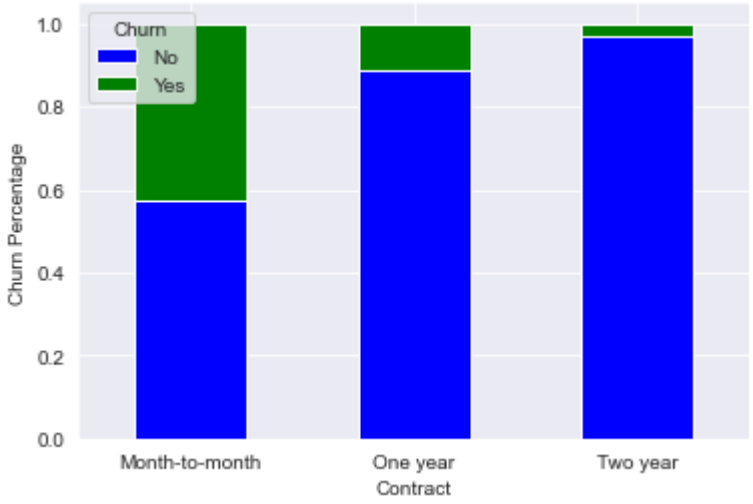

**Internet Service**

In [77]:
```
stacked_plot(df, "InternetService", "Churn")
```



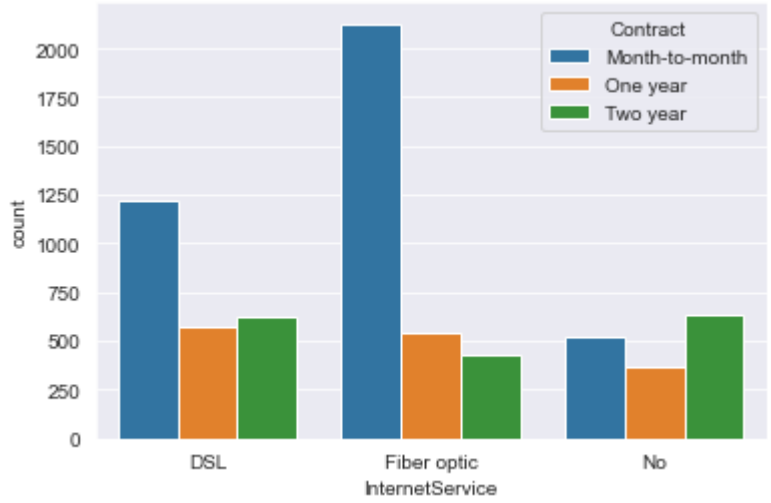

**Contract**

In [78]:
```
stacked_plot(df, "Contract", "Churn")
```



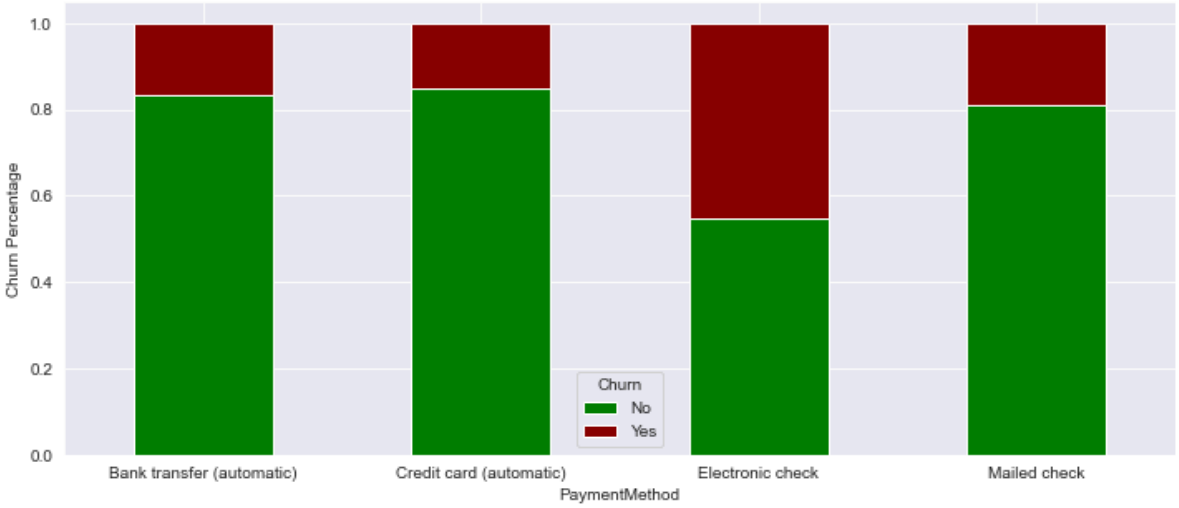
In [79]:
```
sns.countplot(df.InternetService, hue = df.Contract)
```

Out[79]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x277a3a60790>
```



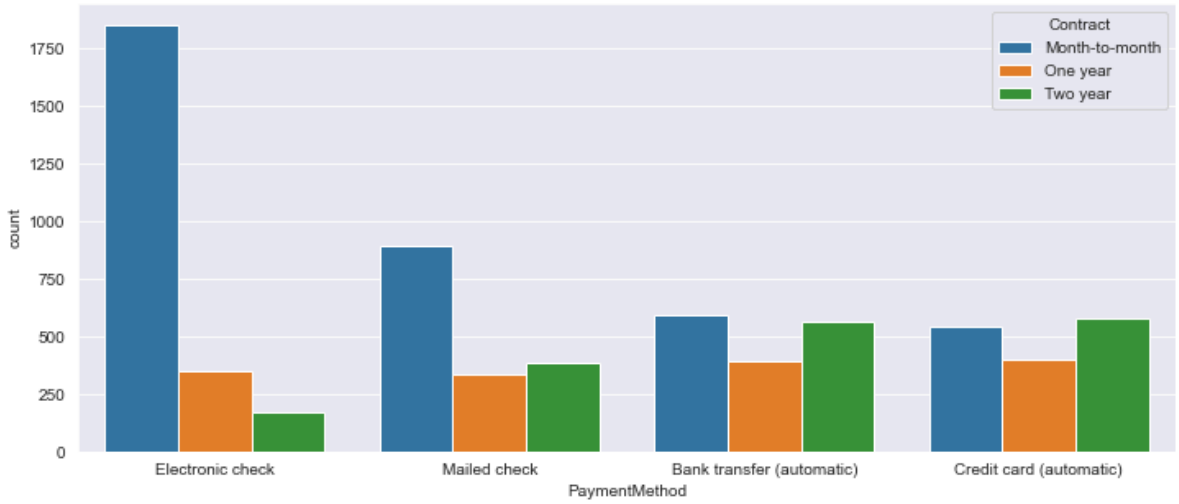

**PaymentMethod**

In [80]:
```
group = "PaymentMethod"
target = "Churn"
fig, ax = plt.subplots(figsize = (12,5))
temp_df = (df.groupby([group, target]).size()/df.groupby(group)[target].count(
temp_df.plot(kind='bar', stacked=True, ax = ax, color = ["green", "darkred"])
ax.xaxis.set_tick_params(rotation=0)
ax.set_xlabel(group)
ax.set_ylabel('Churn Percentage');
```



In [81]:
```
fig, ax = plt.subplots(figsize = (12,5))
sns.countplot(df.PaymentMethod, hue = df.Contract, ax = ax)
```

Out[81]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x277a380ea00>
```




### Label encoding

In [82]:
```
categorical_features = [
    'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
    'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', '
    'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'Payment
]

label_encoders = {}
for col in categorical_features:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
df.head(5)
```

Out[82]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines |
|---|---|---|---|---|---|---|---|---|
| **0** | 7590-VHVEG | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| **1** | 5575-GNVDE | 1 | 0 | 0 | 0 | 34 | 1 | 0 |
| **2** | 3668-QPYBK | 1 | 0 | 0 | 0 | 2 | 1 | 0 |
| **3** | 7795-CFOCW | 1 | 0 | 0 | 0 | 45 | 0 | 1 |
| **4** | 9237-HQITU | 0 | 0 | 0 | 0 | 2 | 1 | 0 |

5 rows × 21 columns

In [83]:
```
# Convert 'TotalCharges' to numeric, coerce errors to NaN
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

df['TotalCharges'].isna().sum()
# Fill missing values in 'TotalCharges' with the median value
df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)
```

In [88]:
```
X = df.drop('Churn', axis=1)
y = df['Churn']
```

In [ ]:
```
# Drop the 'customerID' column
df.drop('customerID', axis=1, inplace=True)
```

In [90]:
```
df.head(5)
```

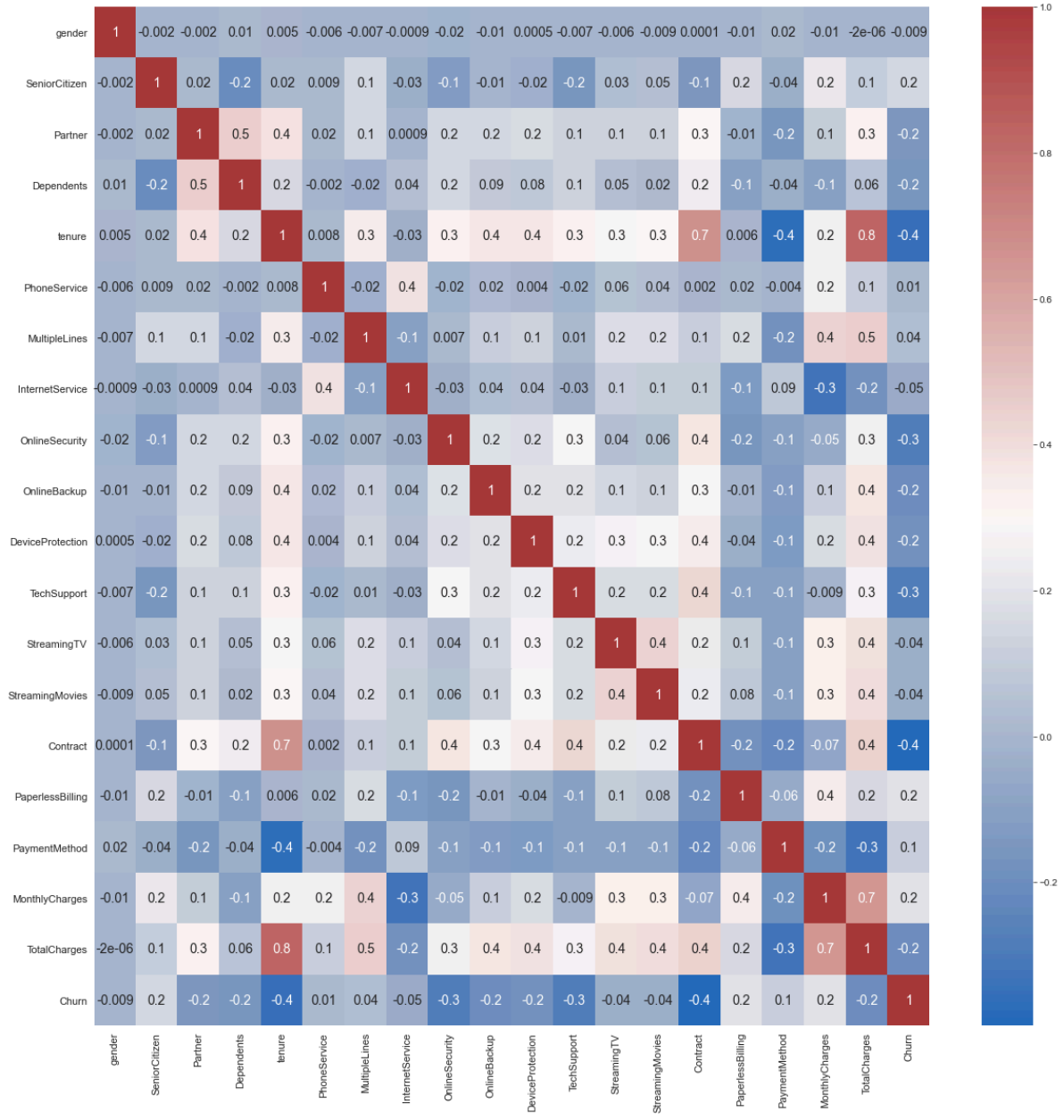
Out[90]:

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetServi |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 | 1 | 0 | 1 | |
| **1** | 1 | 0 | 0 | 0 | 34 | 1 | 0 | |
| **2** | 1 | 0 | 0 | 0 | 2 | 1 | 0 | |
| **3** | 1 | 0 | 0 | 0 | 45 | 0 | 1 | |
| **4** | 0 | 0 | 0 | 0 | 2 | 1 | 0 | |

**Scaling the features**

In [91]:
```
# Scale the features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [92]:
```
axis_labels = ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', '
              'OnlineSecurity','OnlineBackup','DeviceProtection','TechSupport'
fig, ax = plt.subplots(figsize=(20,20))
sns.heatmap(df.corr(), annot=True, ax=ax, cmap = 'vlag', fmt='.1g',
            annot_kws={'fontsize': 14}, xticklabels= axis_labels, yticklabels=
plt.xticks(fontsize=11)
plt.yticks(fontsize=11)
plt.show()
```




In [102]:
```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2
```

**Hyperparameter tuning for Logistic Regression**

In [108]:
```
from sklearn.model_selection import RandomizedSearchCV
log_reg_params = {
    'C': [0.1, 1, 10, 100],
    'solver': ['lbfgs', 'liblinear'],
}

log_reg_random = RandomizedSearchCV(LogisticRegression(max_iter=1000), log_reg
log_reg_random.fit(X_train, y_train)

print("Best parameters for Logistic Regression:", log_reg_random.best_params_)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:278: UserWarning: The total space of parameters 8 is smaller than n_iter=1
0. Running 8 iterations. For exhaustive searches, use GridSearchCV.
  warnings.warn(

Best parameters for Logistic Regression: {'solver': 'lbfgs', 'C': 100}
```

**Hyperparameter tuning for Random Forest Classifier**

In [109]:
```
rf_clf_params = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

rf_clf_random = RandomizedSearchCV(RandomForestClassifier(), rf_clf_params, cv
rf_clf_random.fit(X_train, y_train)

print("Best parameters for Random Forest:", rf_clf_random.best_params_)
```

```
Best parameters for Random Forest: {'n_estimators': 100, 'min_samples_split':
10, 'min_samples_leaf': 2, 'max_depth': None, 'bootstrap': True}
```

**Hyperparameter tuning for Gradient Boosting Classifier**

In [110]:
```
# Hyperparameter tuning for Gradient Boosting Classifier
gb_clf_params = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 0.9, 1.0]
}

gb_clf_random = RandomizedSearchCV(GradientBoostingClassifier(), gb_clf_params
gb_clf_random.fit(X_train, y_train)

print("Best parameters for Gradient Boosting:", gb_clf_random.best_params_)
```

```
Best parameters for Gradient Boosting: {'subsample': 0.9, 'n_estimators': 10
0, 'max_depth': 3, 'learning_rate': 0.1}
```

**Using the best parameters for training and to make predictions**

In [111]:
```
log_reg_best = log_reg_random.best_estimator_
rf_clf_best = rf_clf_random.best_estimator_
gb_clf_best = gb_clf_random.best_estimator_

# Make predictions
log_reg_pred = log_reg_best.predict(X_test)
rf_clf_pred = rf_clf_best.predict(X_test)
gb_clf_pred = gb_clf_best.predict(X_test)

# Evaluate the models using classification report
def print_classification_report(y_test, y_pred, model_name):
    print(f"Classification Report for {model_name}:")
    print(classification_report(y_test, y_pred))

# Print classification reports for each model
print_classification_report(y_test, log_reg_pred, "Logistic Regression")
print_classification_report(y_test, rf_clf_pred, "Random Forest")
print_classification_report(y_test, gb_clf_pred, "Gradient Boosting")
```

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

           0       0.86      0.90      0.88      1036
           1       0.68      0.58      0.62       373

    accuracy                           0.82      1409
   macro avg       0.77      0.74      0.75      1409
weighted avg       0.81      0.82      0.81      1409

Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       0.84      0.92      0.87      1036
           1       0.68      0.50      0.58       373

    accuracy                           0.81      1409
   macro avg       0.76      0.71      0.73      1409
weighted avg       0.80      0.81      0.80      1409

Classification Report for Gradient Boosting:
              precision    recall  f1-score   support

           0       0.85      0.91      0.88      1036
           1       0.68      0.54      0.60       373

    accuracy                           0.81      1409
   macro avg       0.76      0.72      0.74      1409
weighted avg       0.80      0.81      0.80      1409
```

In [ ]:

In [ ]: