NAME: SAJAL CHOURASIA
REG NO: 12108826
Assignment - Predicting Customer Churn in a Telecommunications Company

Understanding Our Dataset:

- Customer ID: Unique identifier for each individual customer.
- gender: Indicates whether the customer is male or female.
- Senior Citizen: Status indicating if the customer is senior.
- Partner: Shows if the customer has a partner.
- Dependents: Indicates if the customer has any dependents.
- tenure: Duration in months with the company.
- Phone Service: Indicates availability of phone service for customer.
- Multiple Lines: Shows if customer has multiple phone lines.
- Internet Service: Type of internet service subscribed by customer.
- Online Security: Indicates if customer has online security service.
- Online Backup: Shows if customer uses online backup service.
- Device Protection: Indicates if customer has device protection service.
- Tech Support: Shows if customer uses tech support service.
- Streaming TV: Indicates if customer has streaming TV service.
- Streaming Movies: Shows if customer has streaming movies service.
- Contract: Type of contract customer has with company.
- Paperless Billing: Indicates if billing is paperless for customer.
- Payment Method: The method customer uses for payment.
- Monthly Charges: The charges customer pays monthly.
- Total Charges: The total charges incurred by customer.
- Churn: Indicates if the customer has churned.

IMPORTING SOME LIBRARIES:

Pandas- Pandas is an open-source library in Python that is made for working with relational or labelled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series.

Matplotlib- Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. One of the greatest benefits of visualization is that it allows us visual access to vast amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

NumPy (Numerical Python)- is a fundamental library for numerical computations in Python. It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

Seaborn- is a Python data visualization library based on Matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. Seaborn is designed to make it easier to create visually appealing and complex plots.

1. Data Cleaning & Preparation:

- Clean the provided data sets to ensure there are no missing or inconsistent values.
- Prepare the data for analysis by merging and aligning the date ranges across all campaigns and ad sets

IMPORTING THE DATASET:

```
df=pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

Let's have information about the dataset:

```
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtectior
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No

Check for Inconsistent Data: Checked both Data Frames for inconsistent data (missing values) by using the `. isnull().sum()` function and duplicate values by using the duplicated().sum()

```
#show if there are full duplicates
df.duplicated().sum()
```

```
22
```

- there are duplicates i will drop them

```
# Drop full duplicates
df.drop_duplicates(inplace=True)
```

```
nans = df.isna().sum().sort_values(ascending=False)
pct = 100 * nans / df.shape[0]
nan_stats = pd.concat([nans, pct], axis=1)
nan_stats.columns = ['num_of_nans', 'percentage_of_nans']
nan_stats
```

	num_of_nans	percentage_of_nans
TotalCharges	11	0.156673
gender	0	0.000000
SeniorCitizen	0	0.000000
MonthlyCharges	0	0.000000
PaymentMethod	0	0.000000
PaperlessBilling	0	0.000000
Contract	0	0.000000
StreamingMovies	0	0.000000
StreamingTV	0	0.000000
TechSupport	0	0.000000
DeviceProtection	0	0.000000
OnlineBackup	0	0.000000
OnlineSecurity	0	0.000000
InternetService	0	0.000000
MultipleLines	0	0.000000
PhoneService	0	0.000000
tenure	0	0.000000
Dependents	0	0.000000
Partner	0	0.000000
Churn	0	0.000000

- TotalCharges column has 11 nans , it is a small number so i will drop them

```
# summary statistics of the data
df.describe()
```

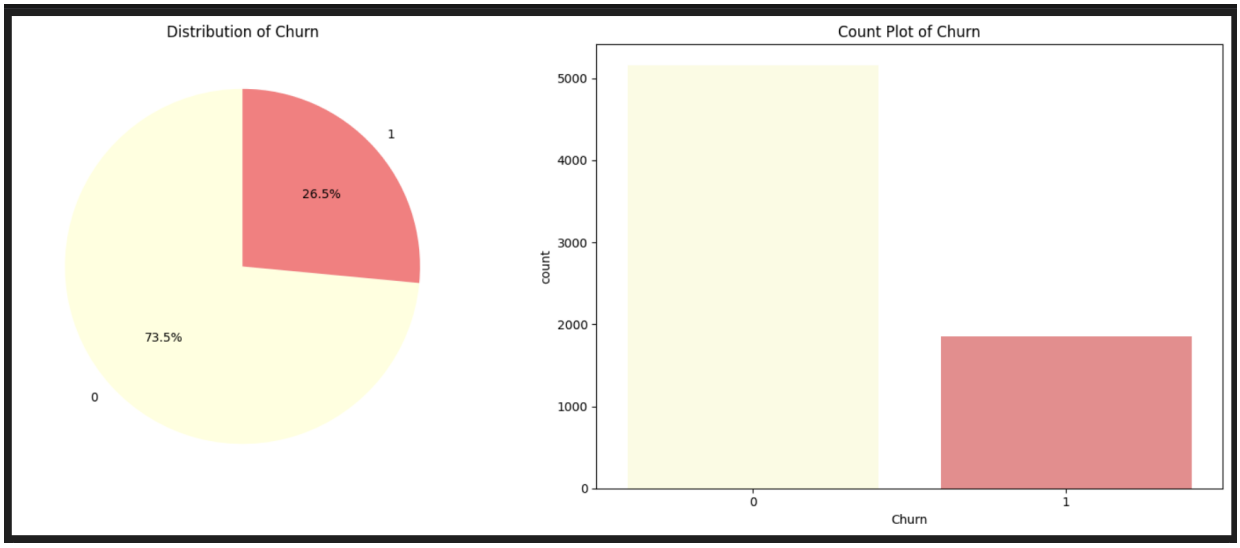
	tenure	MonthlyCharges	TotalCharges
count	7010.000000	7010.000000	7010.000000
mean	32.520399	64.888666	2290.353388
std	24.520441	30.064769	2266.820832
min	1.000000	18.250000	18.800000
25%	9.000000	35.750000	408.312500
50%	29.000000	70.400000	1403.875000
75%	56.000000	89.900000	3807.837500
max	72.000000	118.750000	8684.800000

```
# info about categorical variables
df.describe(include="category")
```

	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport
count	7010	7010	7010	7010	7010	7010	7010	7010	7010	7010	7010
unique	2	2	2	2	2	3	3	3	3	3	3
top	Male	0	No	No	Yes	No	Fiber optic	No	No	No	No
freq	3535	5869	3617	4911	6330	3363	3090	3489	3079	3086	3464

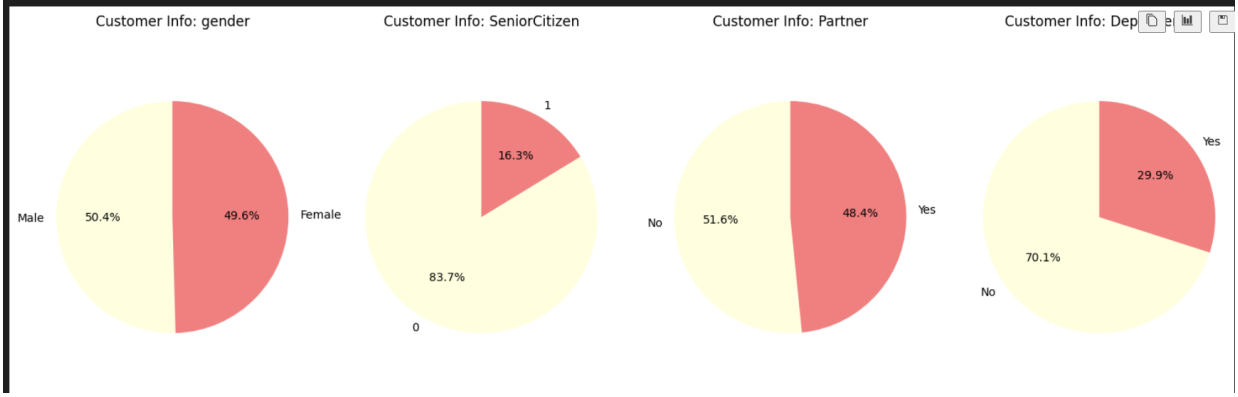
2. Exploratory Data analysis (EDA):

```
# Define custom colors
light_yellow_color = '#FFFFE0'
light_coral_color = '#F08080'
light_salmon_color = '#FFA07A'
light_sea_green_color = '#20B2AA'

# Create a color palette for pie charts
palette = [light_yellow_color, light_coral_color]

# Distribution of Churn
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Assume df is your DataFrame
status_counts = df['Churn'].value_counts()

# Pie chart
axes[0].pie(status_counts, labels=status_counts.index, autopct='%1.1f%%', startangle=90, colors=palette)
axes[0].set_title('Distribution of Churn')

# Count plot
sns.countplot(x='Churn', data=df, palette=palette, ax=axes[1])
axes[1].set_title('Count Plot of Churn')

plt.tight_layout()
plt.show()
```



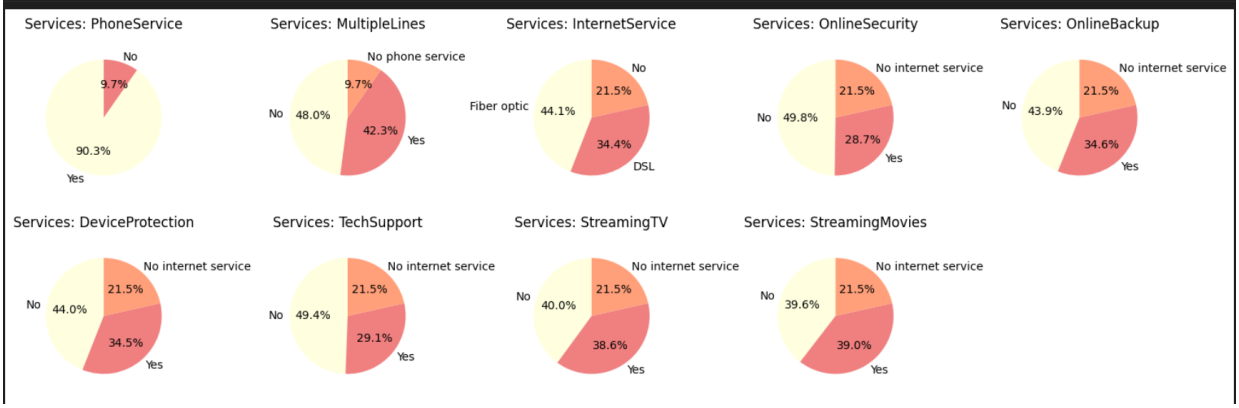
```python
# Plot customer info columns
plt.figure(figsize=(15, 5))
for i, col in enumerate(customer_info, 1):
    plt.subplot(1, 4, i)
    counts = df[col].value_counts(normalize=True)
    plt.pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=90, colors=palette)
    plt.title(f'Customer Info: {col}')
    plt.axis('equal')
plt.tight_layout()
plt.show()
```

✓ 0.2s



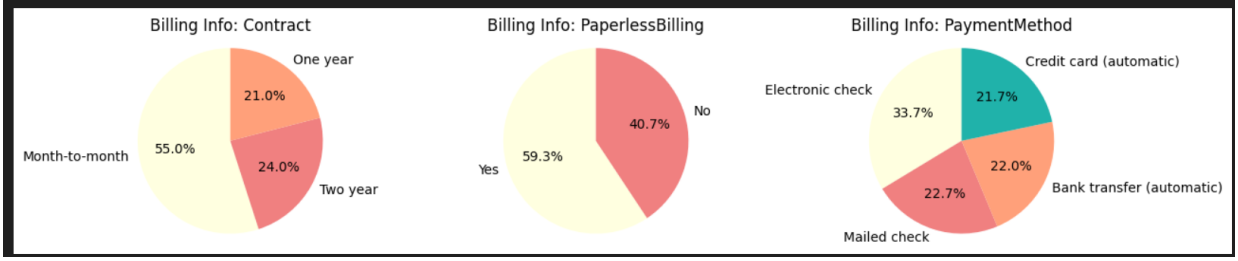

```python
# Plot services columns
plt.figure(figsize=(15, 5))
for i, col in enumerate(services, 1):
    plt.subplot(2, 5, i)
    counts = df[col].value_counts(normalize=True)
    plt.pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=90, colors=palette)
    plt.title(f'Services: {col}')
    plt.axis('equal')
plt.tight_layout()
plt.show()
```

✓ 0.4s



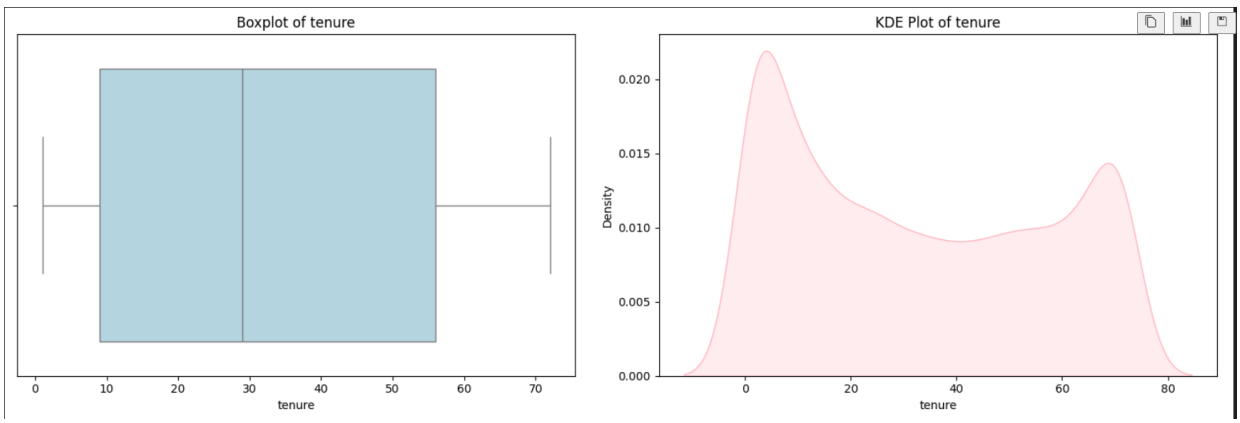
```python
# Plot billing info columns
plt.figure(figsize=(15, 5))
for i, col in enumerate(billing_info, 1):
    plt.subplot(2, 4, i)
    if col in ['MonthlyCharges', 'TotalCharges', 'tenure']:
        sns.histplot(df[col], kde=True, color=light_blue_color)
        plt.title(f'Billing Info: {col}')
    else:
        counts = df[col].value_counts(normalize=True)
        plt.pie(counts, labels=counts.index, autopct='%1.1f%%', startangle=90, colors=palette)
        plt.title(f'Billing Info: {col}')
        plt.axis('equal')
plt.tight_layout()
plt.show()
```

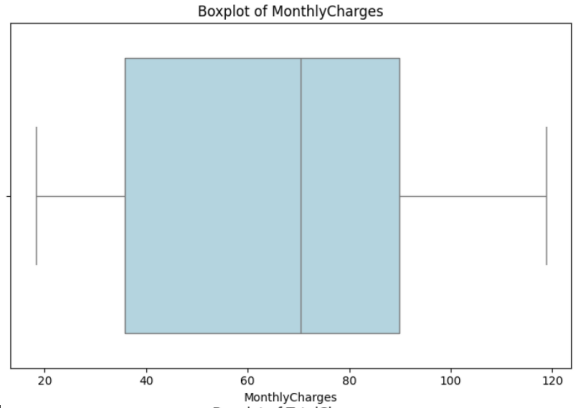
✓ 0.3s

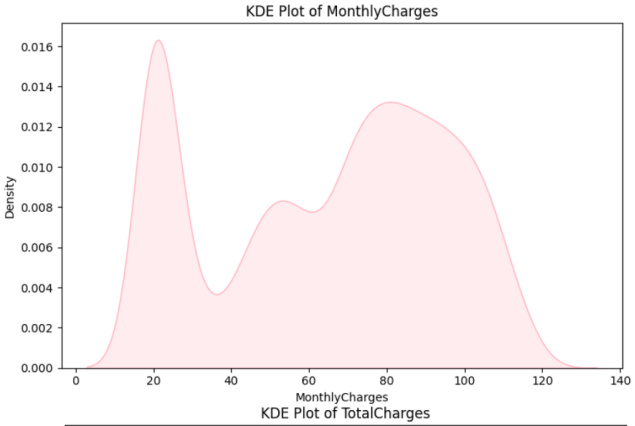


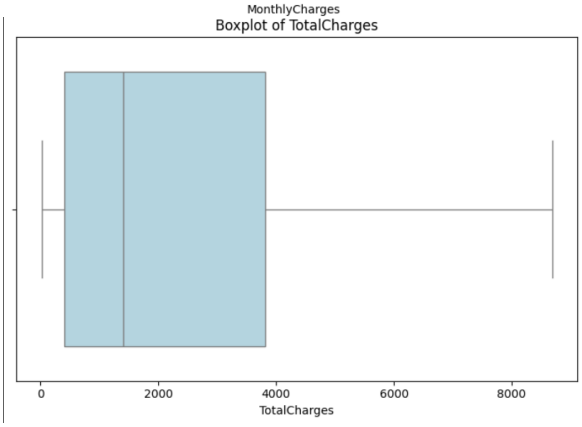

```python
# Define numeric columns
numeric_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']

num_rows = len(numeric_cols)
num_cols = 2

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 5*num_rows))

if num_rows == 1:
    axes = axes.reshape(1, -1)
for i, column in enumerate(numeric_cols):
    # Box plot
    sns.boxplot(x=df[column], ax=axes[i, 0], color=light_blue_color)
    axes[i, 0].set_title(f'Boxplot of {column}')
    axes[i, 0].set_xlabel(column)
    # KDE plot
    sns.kdeplot(data=df[column], ax=axes[i, 1], color=light_pink_color, fill=True)
    axes[i, 1].set_title(f'KDE Plot of {column}')
    axes[i, 1].set_xlabel(column)

plt.tight_layout()
plt.show()
```

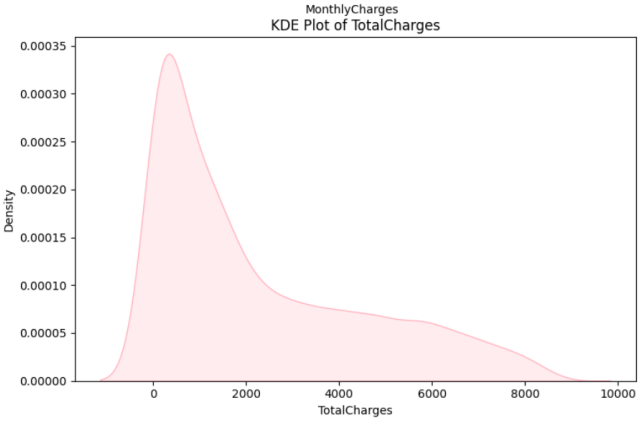
✓ 0.8s





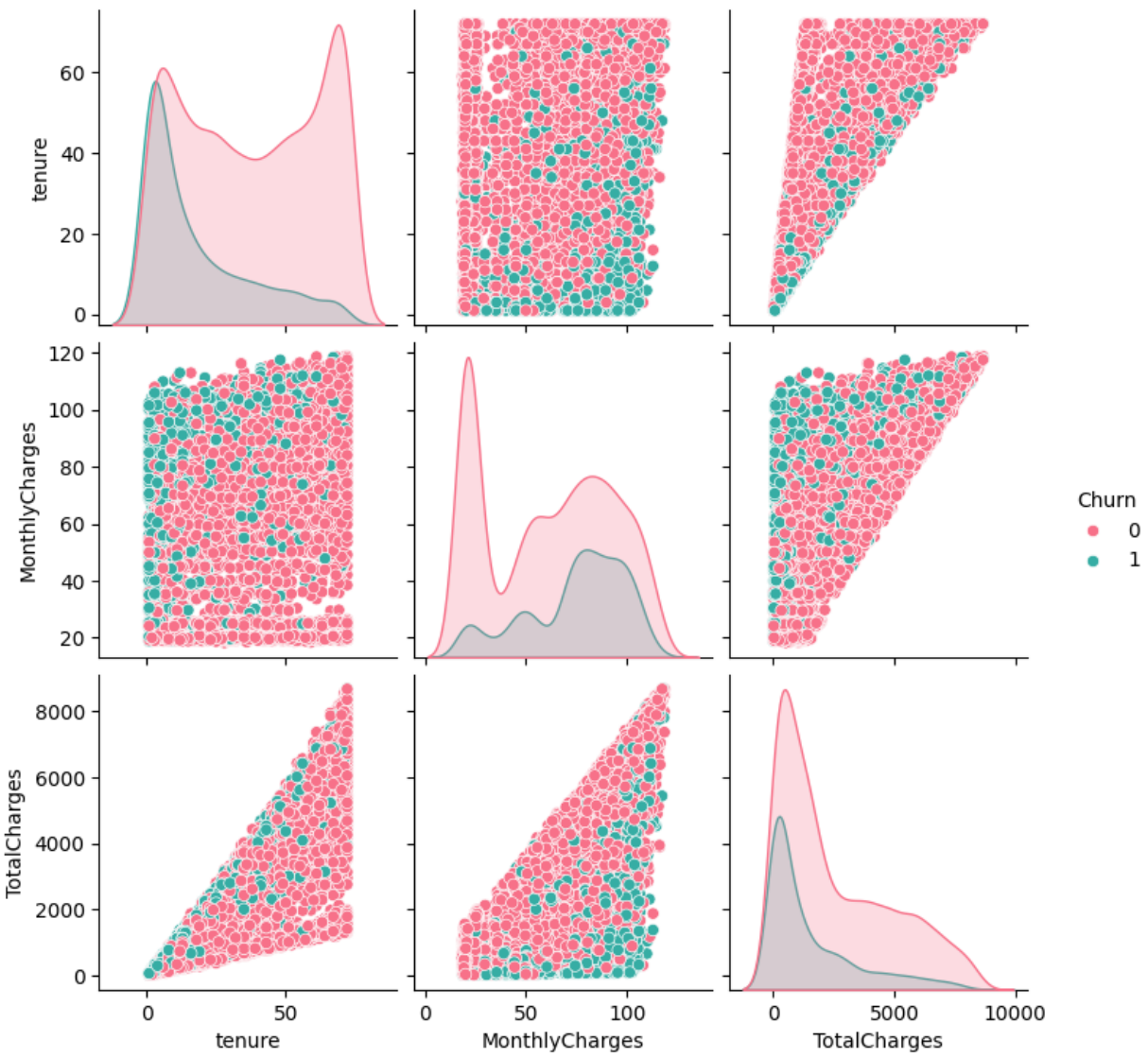







```
sns.pairplot(df, hue='Churn', palette='husl')

plt.show()
```

✓ 3.5s


tenure
MonthlyCharges
TotalCharges
Churn
0
1
0
20
40
60
20
40
60
80
100
120
0
2000
4000
6000
8000
0
50
0
50
100
0
5000
10000

DERIVATION FROM THESE GRAPHS:

1. **Distribution of Churn:**

- A pie chart and a count plot are used to visualize the distribution of churn status.

2. **Categorical Customer Information:**

- This provides a visual summary of the distribution of these attributes among customers.

- For example, we can see the proportion of male vs. female customers, senior citizens, customers with partners, and customers with dependents.

3. **Services Subscriptions:**

- These visualizations show the subscription rates for each service, helping to identify the popularity of various services.

- This can reveal which services are more commonly used and which might need more attention or marketing efforts.

4. **Billing Information:**

- This helps in understanding the different types of contracts customers have, their preference for paperless billing, and the distribution of different payment methods.

- Histograms provide insights into the distribution of monthly charges, total charges, and tenure, revealing potential patterns in customer billing.

5. **Numerical Data Distribution:**

- Box plots show the spread, central tendency, and potential outliers in the data.

- KDE plots help understand the density and distribution shape of these numerical features.

6. **Relationships between Features:**

- This allows for the exploration of pairwise relationships and how they might differ based on whether the customer churned or not.

- The pair-plot can help identify correlations between features and how they relate to churn.

TO REDUCE CHURN WE HAVE TO:

Reducing customer churn is a critical goal for many businesses, as retaining existing customers is often more cost-effective than acquiring new ones.

Improve Customer Service: Providing excellent customer service is crucial for retaining customers. Addressing customer issues promptly and satisfactorily can significantly reduce churn.

Enhance Product/Service Quality: Continuously strive to improve the quality of your products or services. Offering high-quality products/services that meet or exceed customer expectations can increase customer satisfaction and loyalty.

3. MODELLING:

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline, make_pipeline
from imblearn.pipeline import Pipeline as ImbPipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import  f1_score, accuracy_score
from sklearn.metrics import precision_score, recall_score
```

```
# Define classifiers/models
classifiers = [
    ('logreg', LogisticRegression(max_iter=1000)),
    ('rf', RandomForestClassifier()),
    ('gbc', GradientBoostingClassifier())
]
```

```
# Feature selection and Voting Classifier
voting_clf = VotingClassifier(estimators=classifiers, voting='soft')
```

```
# Define features (X) and target variable (y)
X = df.drop(columns=['Churn'])
y = df['Churn']
```

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Pipeline with SMOTE
pipeline = ImbPipeline([
    ('preprocessor', preprocessor),
    ('smote', SMOTE()),
    ('voting_clf', voting_clf)
])
```

```
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='roc_auc')
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
                                        ColumnTransformer(transformers=[('num',
                                                                         Pipeline(steps=[('imputer',
                                                                                          SimpleImputer(strategy='median')),
                                                                                         ('scaler',
                                                                                          StandardScaler())]),
                                                                         ['tenure',
                                                                          'MonthlyCharges',
                                                                          'TotalCharges']),
                                                                        ('cat',
                                                                         Pipeline(steps=[('imputer',
                                                                                          SimpleImputer(strategy='most_frequen
t')),
                                                                                         ('onehot',
                                                                                          OneHotEncoder(handle_unknown='i...
                                        ('smote', SMOTE()),
                                        ('voting_clf',
                                         VotingClassifier(estimators=[('logreg',
                                                                       LogisticRegression(max_iter=1000)),
                                                                      ('rf',
                                                                       RandomForestClassifier()),
                                                                      ('gbc',
                                                                       GradientBoostingClassifier())],
                                                          voting='soft'))]),
             param_grid={'voting_clf__gbc__learning_rate': [0.01, 0.1, 0.2],
                         'voting_clf__logreg__C': [0.1, 1.0, 10],
                         'voting_clf__rf__n_estimators': [50, 100, 200]},
             scoring='roc_auc')
```

```
# Best parameters and estimator
print("Best parameters found: ", grid_search.best_params_)
best_estimator = grid_search.best_estimator_
```

```
Best parameters found:  {'voting_clf__gbc__learning_rate': 0.1, 'voting_clf__logreg__C': 10, 'voting_clf__rf__n_estimators': 10
0}
```

It encompasses preprocessing steps like missing value imputation, feature scaling, and categorical encoding within a pipeline structure. Moreover, class imbalance is addressed through SMOTE oversampling. Hyperparameter tuning via grid search is employed to optimize the classifiers, selecting parameters like regularization strength for logistic regression, the number of trees for random forest, and learning rate for gradient boosting. Model evaluation is conducted using cross-validation with ROC AUC scoring, ensuring robust performance assessment across multiple train-test splits.

4. EVALUATION ON THE BASIS OF PRECISION, RECALL, F1 SCORE AND ACCURACY:

```
# Best parameters and estimator
print("Best parameters found: ", grid_search.best_params_)
best_estimator = grid_search.best_estimator_
```

```
Best parameters found:  {'voting_clf__gbc__learning_rate': 0.1, 'voting_clf__logreg__C': 10, 'voting_clf__rf__n_estimators': 10
0}
```

```
# Predictions
y_pred = best_estimator.predict(X_test)
```

```
# Evaluation
precision = precision_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
print(f"recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
print(f"precision: {precision:.2f}")
print(f"Accuracy: {accuracy:.2f}")
```

```
recall: 0.70
F1 Score: 0.60
precision: 0.52
Accuracy: 0.78
```