In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from plotnine import ggplot, aes, geom_boxplot, labs, theme, element_t
```

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/My Drive/Colab Data/train.csv')
df.info()
```

```
Mounted at /content/drive
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31480 entries, 0 to 31479
Data columns (total 20 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   id                            31480 non-null  int64
 1   target                        31480 non-null  object
 2   day                           31480 non-null  int64
 3   month                         31480 non-null  object
 4   duration                      31480 non-null  int64
 5   contactId                     31480 non-null  int64
 6   age                           31480 non-null  int64
 7   gender                        31480 non-null  object
 8   job                           31480 non-null  object
 9   maritalStatus                 31480 non-null  object
 10  education                     31480 non-null  object
 11  creditFailure                 31480 non-null  object
 12  accountBalance                31480 non-null  int64
 13  house                         31480 non-null  object
 14  credit                        31480 non-null  object
 15  contactType                   31480 non-null  object
 16  numberOfContacts              31480 non-null  int64
 17  daySinceLastCampaign          5738 non-null   float64
 18  numberOfContactsLastCampaign  31480 non-null  int64
 19  lastCampaignResult            31480 non-null  object
dtypes: float64(1), int64(8), object(11)
memory usage: 4.8+ MB
```

In [3]:
```
df
```

Out[3]:

	id	target	day	month	duration	contactId	age	gender	job	maritalS
0	432148809	no	27	may	166	623	30	female	worker	ma
1	432184318	no	26	oct	183	1992	42	female	manager	ma
2	432182482	no	5	jun	227	2778	26	female	services	s
3	432150520	no	2	jun	31	3070	34	male	unemployed	div
4	432145870	no	15	may	1231	6583	48	male	worker	ma
...	...	...	...	...	...	...	...	...	...	
31475	432184725	yes	30	nov	1628	69542367	58	female	technical	ma
31476	432147139	no	21	may	173	69542565	40	female	manager	s
31477	432166958	no	17	nov	422	69543453	51	female	worker	ma
31478	432166312	no	29	aug	69	69544121	30	male	technical	ma
31479	432171709	no	2	feb	171	69546604	50	male	technical	div

31480 rows × 20 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['id', 'target', 'day', 'month', 'duration', 'contactId', 'age'
,
       'gender', 'job', 'maritalStatus', 'education', 'creditFailure'
,
       'accountBalance', 'house', 'credit', 'contactType', 'numberOfC
ontacts',
       'daySinceLastCampaign', 'numberOfContactsLastCampaign',
       'lastCampaignResult'],
      dtype='object')
```

In [5]:
```
df.columns = ('id', 'purchase', 'day', 'month', 'duration', 'contactIc
       'gender', 'job', 'maritalStatus', 'education', 'creditFailure',
       'accountBalance', 'house', 'credit', 'contactType', 'numberOfCo
       'daySinceLastCampaign', 'numberOfContactsLastCampaign',
       'lastCampaignResult')
```

In [6]:
```
df[df['daySinceLastCampaign'].isnull()]['numberOfContactsLastCampaign'
```

Out[6]:
```
array([0])
```

In [7]:
```
df['daySinceLastCampaign'].fillna(-1, inplace=True)
```

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
id                            0
purchase                      0
day                           0
month                         0
duration                      0
contactId                     0
age                           0
gender                        0
job                           0
maritalStatus                 0
education                     0
creditFailure                 0
accountBalance                0
house                         0
credit                        0
contactType                   0
numberOfContacts              0
daySinceLastCampaign          0
numberOfContactsLastCampaign  0
lastCampaignResult            0
dtype: int64
```

In [9]:
```
encoded_df=pd.get_dummies(data=df, drop_first=True)
```

In [10]:
```
import pandas as pd

# Numeric features
numericFeatures = ['day', 'duration', 'age', 'accountBalance', 'number

# Calculate mean values grouped by 'purchase'
grouped_means = df[numericFeatures].groupby('purchase').mean()

# Display the table
print(grouped_means)

#findings: it shows that longer the call with the potential customer i
#the more money a person has in their account, the more likely they wo
#??? interestingly, the more time has passed since the last campaign b
```

```
                day    duration        age  accountBalance  numberOfC
ontacts  \
purchase
no        15.903384  221.600036  40.829770     1287.468143          2
.866379
yes       15.015405  535.535135  41.731351     1807.032703          2
.128649

          daySinceLastCampaign  numberOfContactsLastCampaign
purchase
no                   36.415551                      0.507847
yes                  68.985676                      1.158649
```

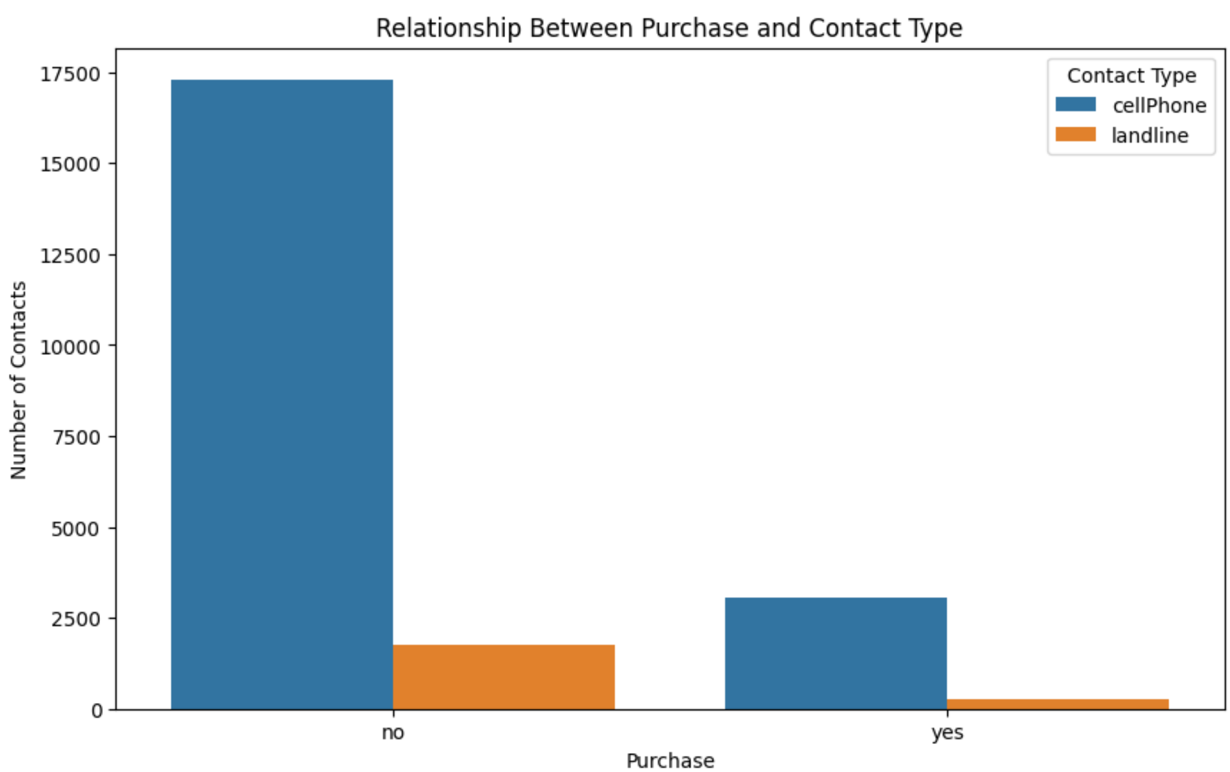
In [11]:

```
import plotly.graph_objects as go

#campaign_success = purchased
#campaign_failure = not purchased

# Data
Labels = ['Campaign_Success', 'Campaign_Failure']
successes = len(df[df['purchase'] == 'yes'])
failures = len(df[df['purchase'] == 'no'])
success_color = 'green'
failure_color = 'red'
Values = [successes, failures]

# Create bar graph
fig = go.Figure(data=[go.Bar(x=Labels, y=Values, marker_color=[success

# Add title and axis labels
fig.update_layout(title='Campaign Success vs Failure Comparison',
                  xaxis_title='Did the Target Consumer Make a Purchase
                  yaxis_title='Count of Consumers')

# Show the graph
fig.show()




#the distribution shows that our current dataset is imbalanced, so we
```

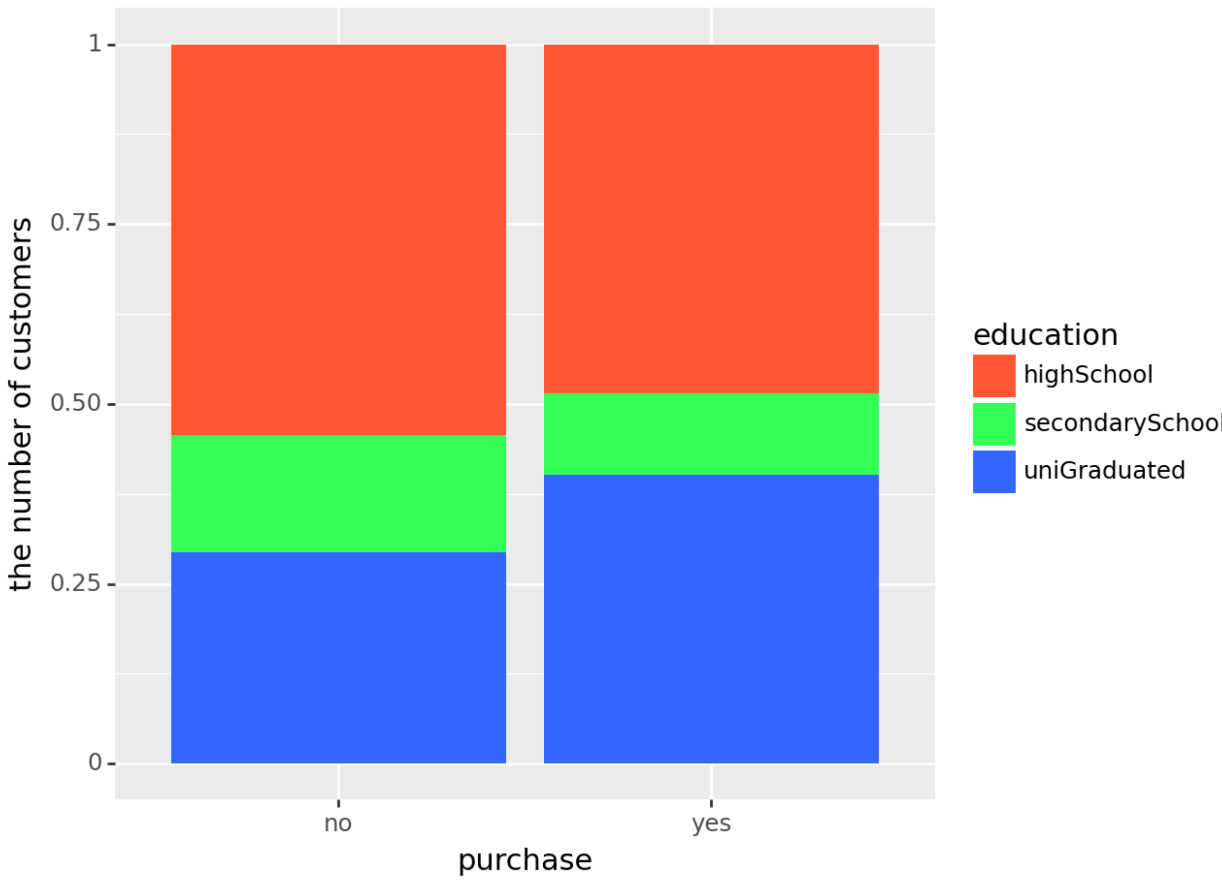
In [12]:

```
df_compare=df.copy()
df_compare.replace("unknown", np.nan, inplace=True)
```

In [13]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Assuming Data is your DataFrame and is already defined.

# Count the occurrences of each combination of 'purchase' and 'contact
purchase_contact_counts = df_compare.groupby(['purchase', 'contactType

plt.figure(figsize=(10, 6))

# Use seaborn to create the bar chart
sns.barplot(data=purchase_contact_counts, x='purchase', y='counts', hu

# Add title and labels to the plot
plt.title('Relationship Between Purchase and Contact Type')
plt.xlabel('Purchase')
plt.ylabel('Number of Contacts')

# Display the plot
plt.legend(title='Contact Type')
plt.show()
```



In [14]:
```
from plotnine import ggplot, aes, geom_bar, scale_fill_manual, labs


df_filtered = df[df['education'] != 'unknown']

(
# Now create the plot with the filtered DataFrame
ggplot(df_filtered) +
    aes(x="purchase", fill="education") +
    geom_bar(position="fill") +
    scale_fill_manual(values=["#FF5733", "#33FF57", "#3366FF"]) +
    labs(y="the number of customers")
)
```



Out[14]: <Figure Size: (640 x 480)>

In [15]:
```
df
```

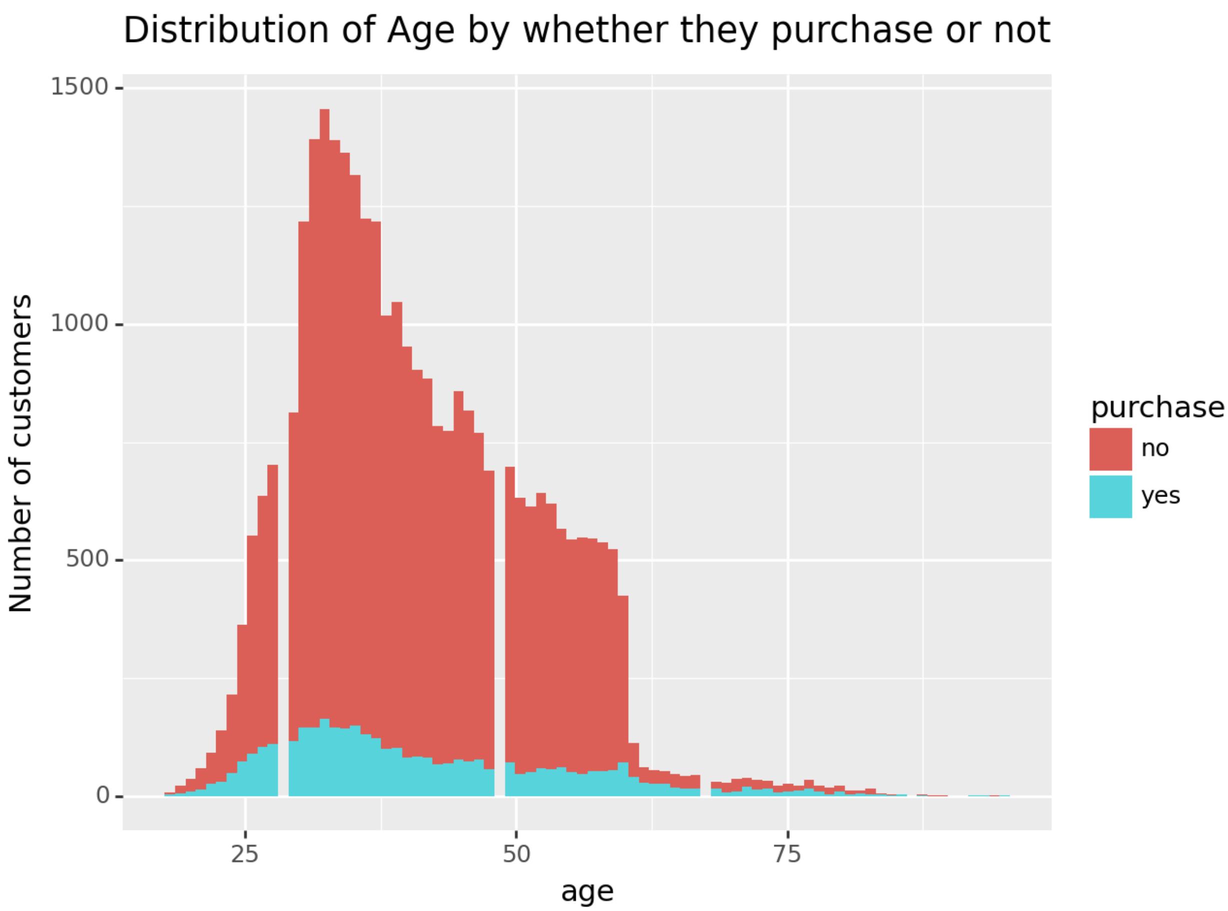
Out[15]:

	id	purchase	day	month	duration	contactId	age	gender	job	marit
0	432148809	no	27	may	166	623	30	female	worker	
1	432184318	no	26	oct	183	1992	42	female	manager	
2	432182482	no	5	jun	227	2778	26	female	services	
3	432150520	no	2	jun	31	3070	34	male	unemployed	
4	432145870	no	15	may	1231	6583	48	male	worker	
...	...	...	...	...	...	...	...	...	...	
31475	432184725	yes	30	nov	1628	69542367	58	female	technical	
31476	432147139	no	21	may	173	69542565	40	female	manager	
31477	432166958	no	17	nov	422	69543453	51	female	worker	
31478	432166312	no	29	aug	69	69544121	30	male	technical	
31479	432171709	no	2	feb	171	69546604	50	male	technical	

31480 rows × 20 columns

In [16]:
```
from plotnine import ggplot, aes, geom_histogram, labs, scale_fill_gra

(
  ggplot(df, aes(x="age", fill="purchase"))+
  geom_histogram()+
  labs(
      y= "Number of customers",
      title= "Distribution of Age by whether they purchase or not"
  )
)
```

```
/usr/local/lib/python3.10/dist-packages/plotnine/stats/stat_bin.py:10
9: PlotnineWarning:

'stat_bin()' using 'bins = 82'. Pick better value with 'binwidth'.
```



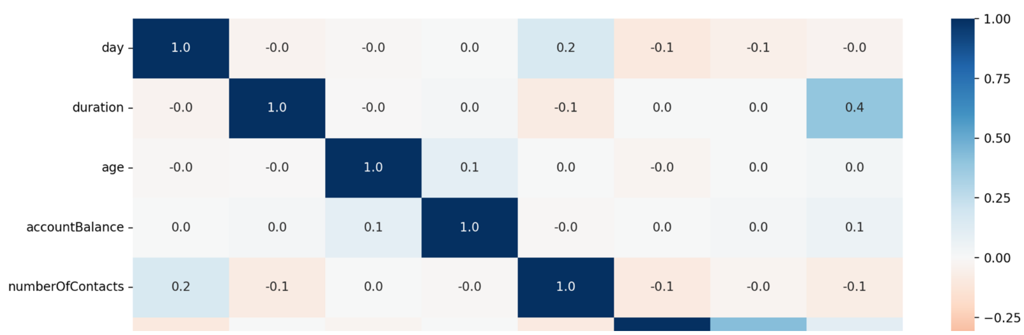
Out[16]: <Figure Size: (640 x 480)>

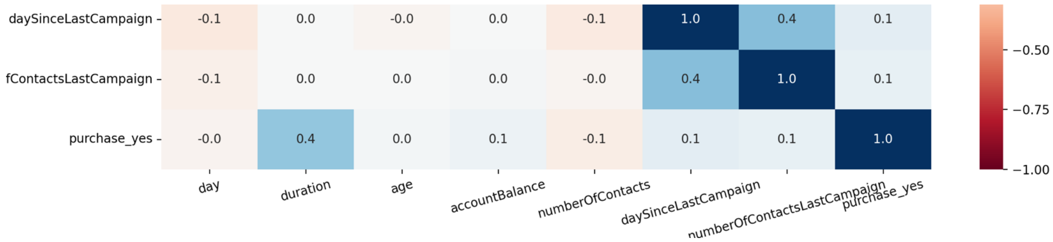
In [17]:

```
import seaborn as sns
encoded_df2=pd.get_dummies(data=df, columns=["purchase"], drop_first=T
dropped_column=encoded_df2[['id','contactId']]
encoded_df2=encoded_df2.drop(dropped_column, axis=1)
print(encoded_df2)
corr_mat=encoded_df2.corr(numeric_only=True)
plt.figure(figsize=(14,7))
sns.heatmap(corr_mat, annot=True, vmin=-1, vmax=1, fmt=".1f", cmap="Rc
plt.xticks(rotation=15)
plt.yticks(rotation=0)
plt.show()
```

```
       day month  duration  age  gender        job maritalStatus  \
0       27   may       166   30  female     worker       married
1       26   oct       183   42  female    manager       married
2        5   jun       227   26  female   services        single
3        2   jun        31   34    male unemployed      divorced
4       15   may      1231   48    male     worker       married
...    ...   ...       ...  ...     ...        ...           ...
31475   30   nov      1628   58  female  technical       married
31476   21   may       173   40  female    manager        single
31477   17   nov       422   51  female     worker       married
31478   29   aug        69   30    male  technical       married
31479    2   feb       171   50    male  technical      divorced

             education creditFailure  accountBalance house credit con
tactType  \
0           highSchool            no            -202    no     no
unknown
1         uniGraduated            no            2463    no     no   c
ellPhone
2           highSchool            no            2158   yes    yes
landline
3         uniGraduated           yes              75   yes     no
unknown
4      secondarySchool            no             559   yes     no
unknown
...                ...           ...             ...   ...    ...
...
31475       highSchool            no            3399    no     no
landline
31476  secondarySchool            no             858   yes     no
unknown
31477       highSchool            no            1414   yes     no
unknown
31478     uniGraduated            no               1    no     no   c
ellPhone
31479       highSchool            no               8    no     no   c
ellPhone
```

```
       numberOfContacts  daySinceLastCampaign  numberOfContactsLastCa
mpaign  \
0                     2                  -1.0
0
1                     2                  -1.0
0
2                     1                  -1.0
0
3                     3                  -1.0
0
4                     2                  -1.0
0
...                 ...                   ...
...
31475                 2                 188.0
8
31476                 1                  -1.0
0
31477                 3                 186.0
2
31478                21                  -1.0
0
31479                 2                   5.0
1

      lastCampaignResult  purchase_yes
0                unknown         False
1                unknown         False
2                unknown         False
3                unknown         False
4                unknown         False
...                  ...           ...
31475            success          True
31476            unknown         False
31477            failure         False
31478            unknown         False
31479              other         False

[31480 rows x 18 columns]
```





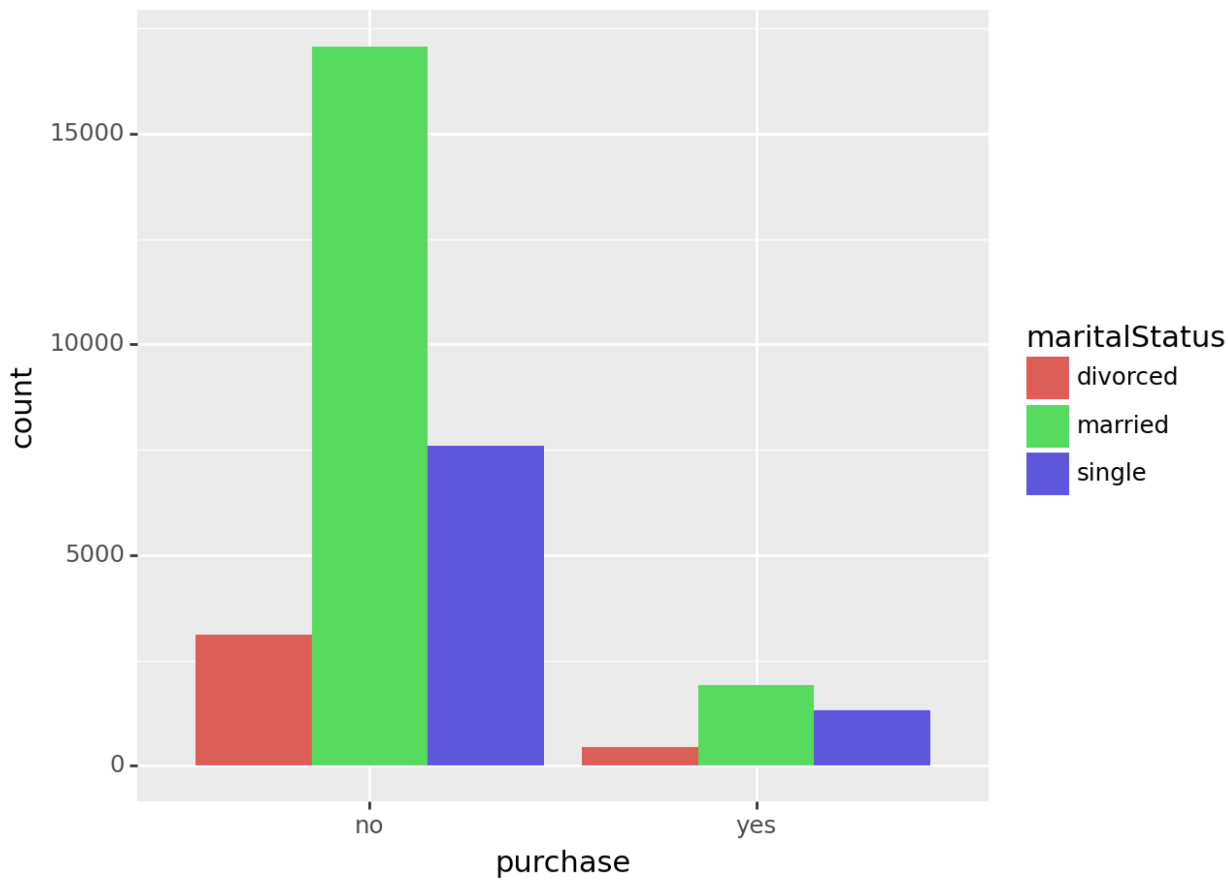
In [18]:
```
encoded_df2
```

Out[18]:

	day	month	duration	age	gender	job	maritalStatus	education	creditF
0	27	may	166	30	female	worker	married	highSchool	
1	26	oct	183	42	female	manager	married	uniGraduated	
2	5	jun	227	26	female	services	single	highSchool	
3	2	jun	31	34	male	unemployed	divorced	uniGraduated	
4	15	may	1231	48	male	worker	married	secondarySchool	
...	...	...	...	...	...	...	...	...	
31475	30	nov	1628	58	female	technical	married	highSchool	
31476	21	may	173	40	female	manager	single	secondarySchool	
31477	17	nov	422	51	female	worker	married	highSchool	
31478	29	aug	69	30	male	technical	married	uniGraduated	
31479	2	feb	171	50	male	technical	divorced	highSchool	

31480 rows × 18 columns

In [19]:
```
(
    ggplot(df, aes(x="purchase"))+
          geom_bar(aes(fill="maritalStatus"), position="dodge")
)
```



Out[19]: <Figure Size: (640 x 480)>

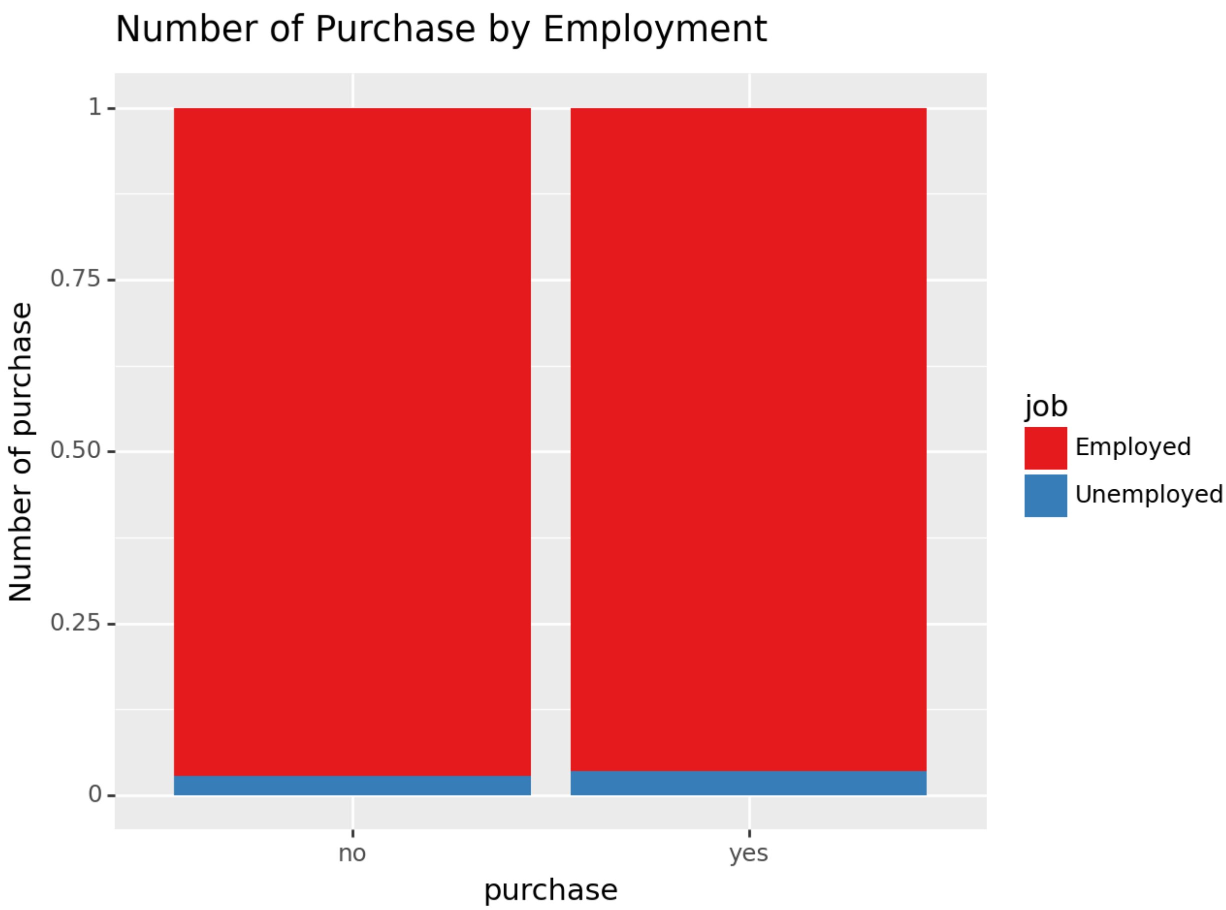
In [20]:
```
df_job=df.copy()
df_job["job"]=df_job['job'].apply(lambda x: 'Unemployed' if x == 'unem
df_job
```

Out[20]:

	id	purchase	day	month	duration	contactId	age	gender	job	mari
0	432148809	no	27	may	166	623	30	female	Employed	
1	432184318	no	26	oct	183	1992	42	female	Employed	
2	432182482	no	5	jun	227	2778	26	female	Employed	
3	432150520	no	2	jun	31	3070	34	male	Unemployed	
4	432145870	no	15	may	1231	6583	48	male	Employed	
...	...	...	...	...	...	...	...	...	...	
31475	432184725	yes	30	nov	1628	69542367	58	female	Employed	
31476	432147139	no	21	may	173	69542565	40	female	Employed	
31477	432166958	no	17	nov	422	69543453	51	female	Employed	
31478	432166312	no	29	aug	69	69544121	30	male	Employed	
31479	432171709	no	2	feb	171	69546604	50	male	Employed	

31480 rows × 20 columns

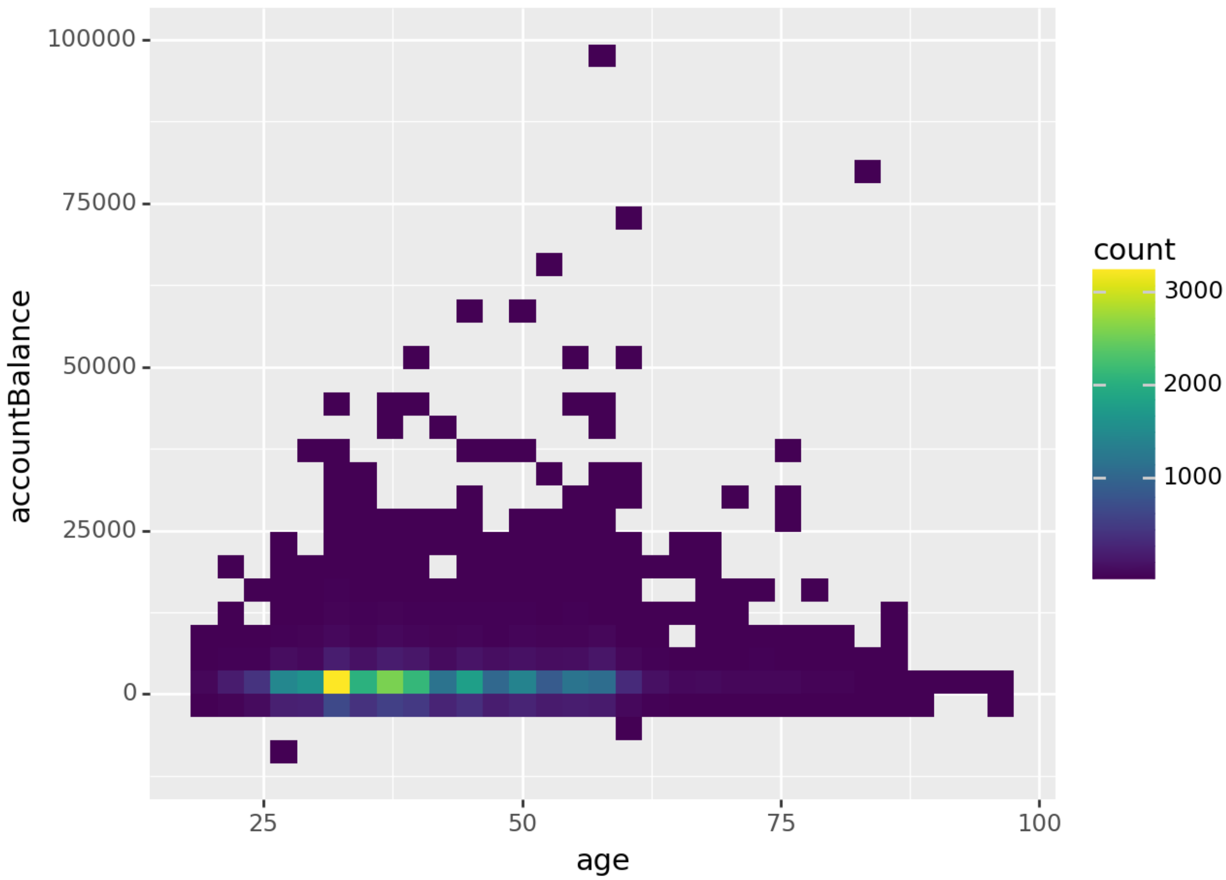
In [21]:
```
from plotnine import scale_fill_brewer, scale_color_brewer
(
    ggplot(df_job, aes(x="purchase", fill="job"))+
    geom_bar(position="fill")+
    labs(
        y="Number of purchase",
         title="Number of Purchase by Employment")+
    scale_fill_brewer(type="qual", palette="Set1")
)
```



Out[21]: <Figure Size: (640 x 480)>

In [21]:

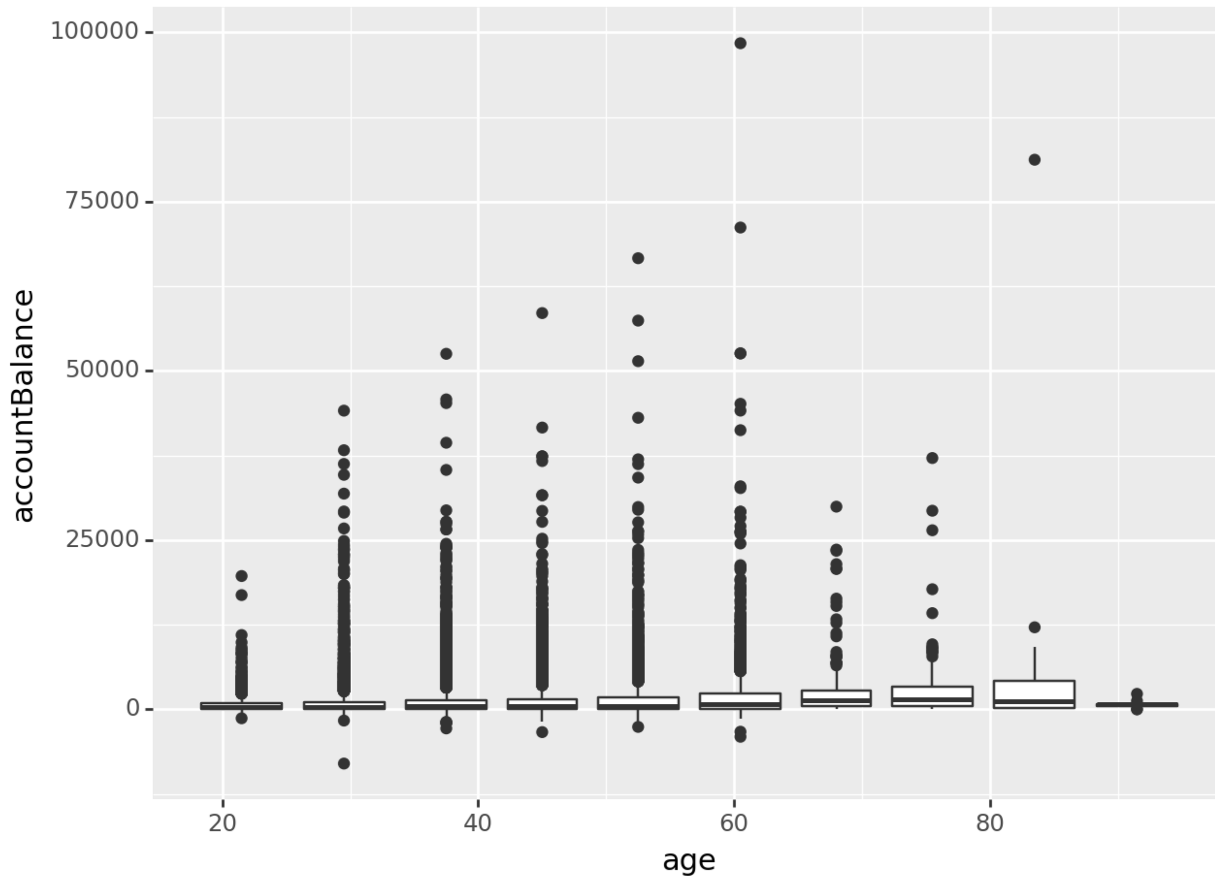
In [22]:
```
# Correlation between Age and AccountBalance
from plotnine import scale_fill_brewer, scale_color_brewer, geom_bin2d

(
    ggplot(df, aes(x="age", y="accountBalance"))+
        geom_bin2d()
)
```



Out[22]: <Figure Size: (640 x 480)>

In [23]:

```
df_age_by_10=df.copy()
df_age_by_10['cut_age']=pd.cut(df_age_by_10['age'], bins=10, labels=Fa

(
    ggplot(df_age_by_10, aes(x="age", y="accountBalance"))+
          geom_boxplot(aes(group='cut_age'))

)
```

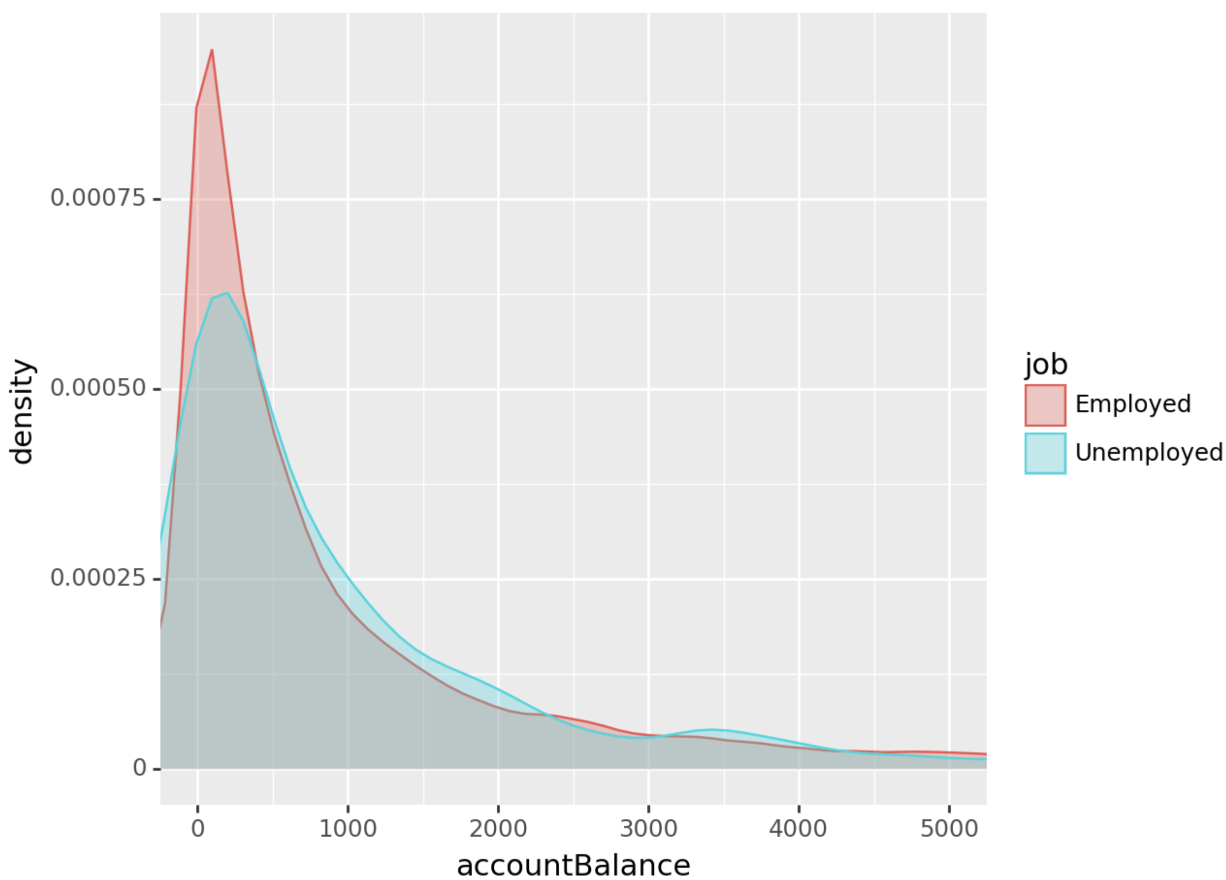


Out[23]: <Figure Size: (640 x 480)>

In [23]:

In [23]:

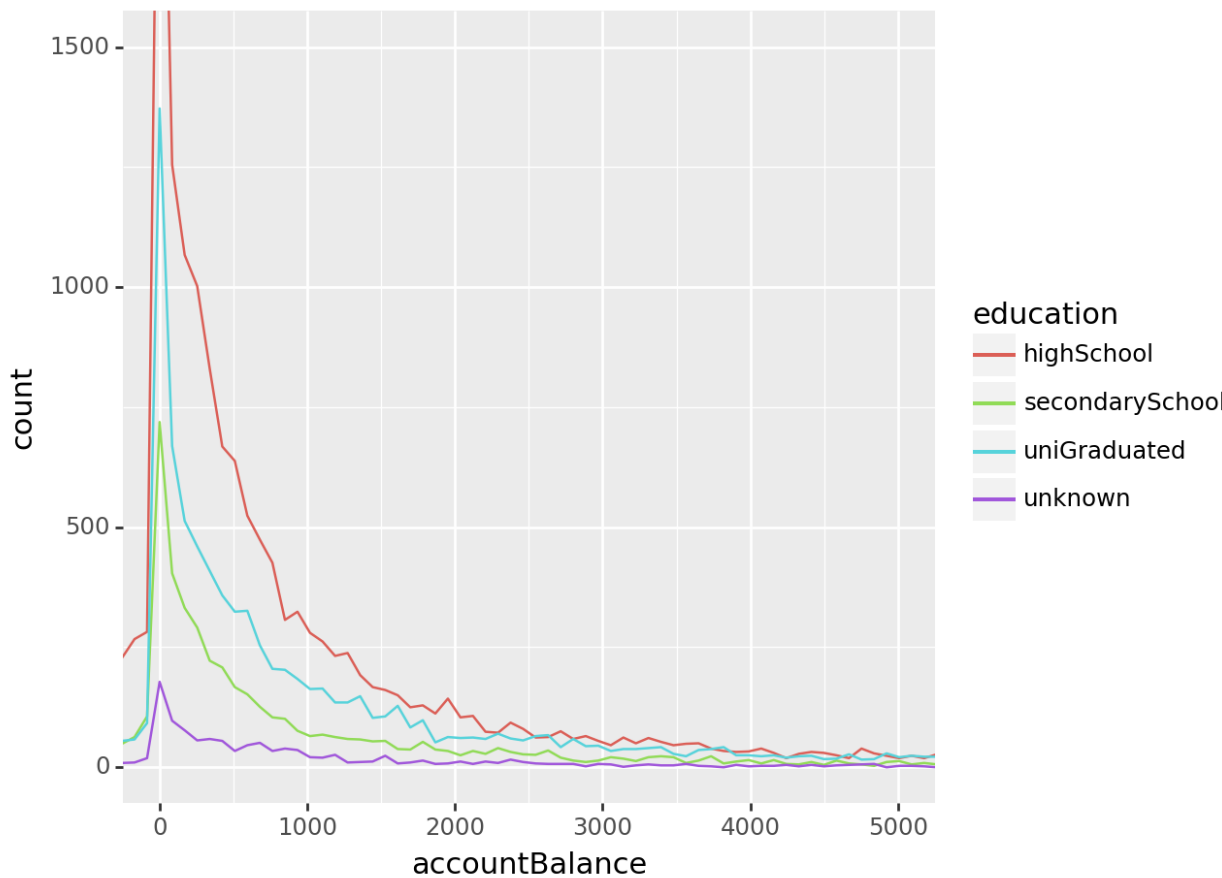
In [24]:
```
from plotnine import scale_fill_brewer, scale_color_brewer, geom_freqp
from plotnine.facets import facet_grid, facet_wrap


p1=(
    ggplot(df_job, aes(x="accountBalance", color="job",fill="job"))+
        geom_density(alpha=0.3)+
        coord_cartesian(xlim = [0, 5000])

 )



p1
```



Out[24]: <Figure Size: (640 x 480)>

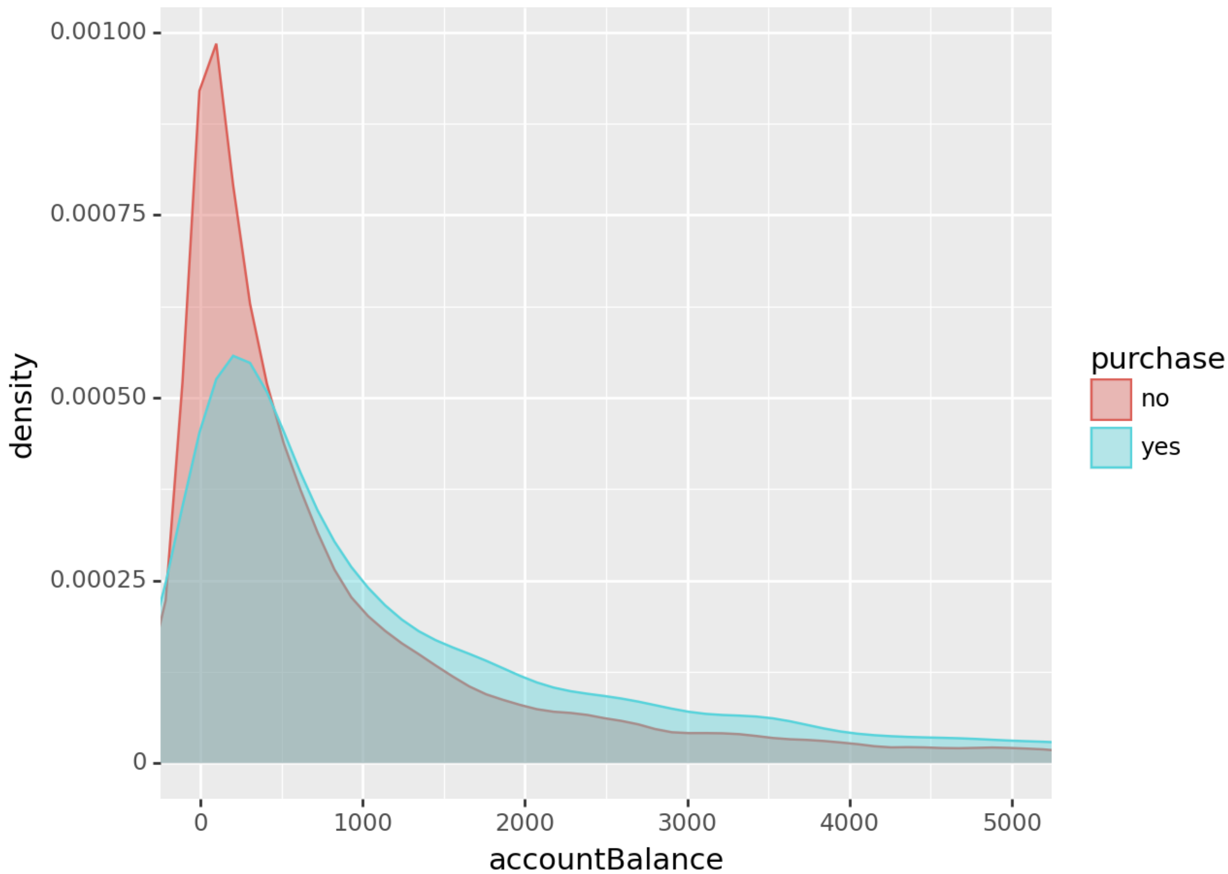
In [25]:
```
p2= (ggplot(df)+
    geom_freqpoly(aes(x="accountBalance", color="education",fill="educ
    coord_cartesian(xlim = [0, 5000], ylim=[0,1500])
)
p2
```

```
/usr/local/lib/python3.10/dist-packages/plotnine/stats/stat_bin.py:10
9: PlotnineWarning:

'stat_bin()' using 'bins = 1255'. Pick better value with 'binwidth'.
```



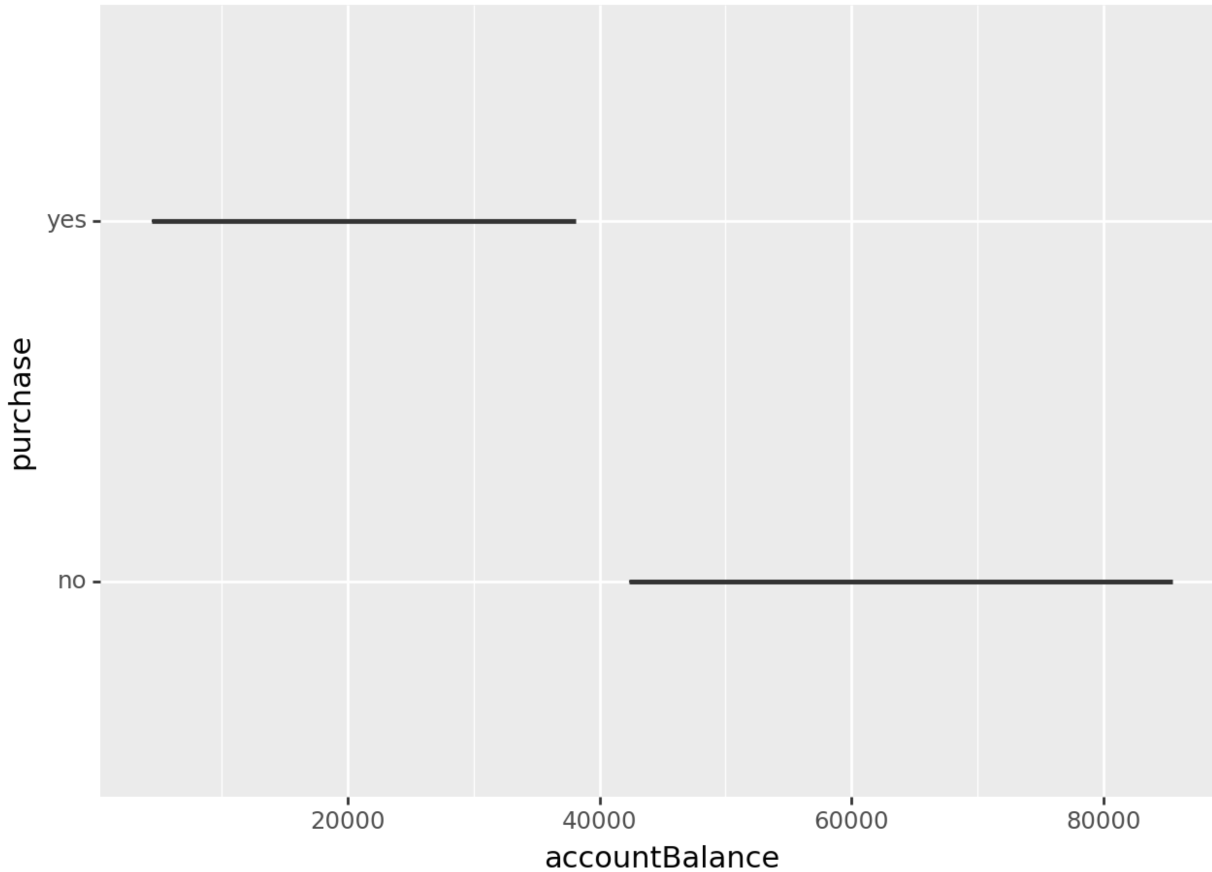
Out[25]: <Figure Size: (640 x 480)>

In [26]:
```
(
    ggplot(df, aes(x="accountBalance", color="purchase", fill="purchas
          geom_density(alpha=0.4)+
          coord_cartesian(xlim = [0, 5000])
)
```



Out[26]: <Figure Size: (640 x 480)>

In [27]:
```
(
    ggplot(df)+
        geom_boxplot(aes(x="accountBalance", y='purchase'))
)
```



Out[27]: <Figure Size: (640 x 480)>

In [28]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Libraries for Data preprocessing
from sklearn.preprocessing import OneHotEncoder
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#Machine learning
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier


#Libraries for machine learning Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve,auc
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

#Code for Neural Network
from keras.models import Sequential
from keras.layers import Dense
```

In [29]:
```
df.copy()
data=df.drop(["daySinceLastCampaign","lastCampaignResult","id","contac
```

In [30]:
```
data
```

Out[30]:

	purchase	day	month	duration	age	gender	job	maritalStatus	educati
0	no	27	may	166	30	female	worker	married	highSch
1	no	26	oct	183	42	female	manager	married	uniGraduat
2	no	5	jun	227	26	female	services	single	highSch
3	no	2	jun	31	34	male	unemployed	divorced	uniGraduat
4	no	15	may	1231	48	male	worker	married	secondarySch
...	...	...	...	...	...	...	...	...	
31475	yes	30	nov	1628	58	female	technical	married	highSch
31476	no	21	may	173	40	female	manager	single	secondarySch
31477	no	17	nov	422	51	female	worker	married	highSch
31478	no	29	aug	69	30	male	technical	married	uniGraduat
31479	no	2	feb	171	50	male	technical	divorced	highSch

31480 rows × 16 columns

In [31]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31480 entries, 0 to 31479
Data columns (total 16 columns):
 #   Column                        Non-Null Count  Dtype 
---  ------                        --------------  ----- 
 0   purchase                      31480 non-null  object
 1   day                           31480 non-null  int64 
 2   month                         31480 non-null  object
 3   duration                      31480 non-null  int64 
 4   age                           31480 non-null  int64 
 5   gender                        31480 non-null  object
 6   job                           31480 non-null  object
 7   maritalStatus                 31480 non-null  object
 8   education                     31480 non-null  object
 9   creditFailure                 31480 non-null  object
 10  accountBalance                31480 non-null  int64 
 11  house                         31480 non-null  object
 12  credit                        31480 non-null  object
 13  contactType                   31480 non-null  object
 14  numberOfContacts              31480 non-null  int64 
 15  numberOfContactsLastCampaign  31480 non-null  int64 
dtypes: int64(6), object(10)
memory usage: 3.8+ MB
```

In [32]:
```
purchase_new = {"purchase": {"yes": 1, "no": 0}}
data.replace(purchase_new, inplace=True)
```

In [33]:
```
# Selecting all Categorical Features
category = data.select_dtypes(include=["object"])

enc = OneHotEncoder(sparse=False)

# Applying OneHotEncoder to the categorical data
Cat_new = enc.fit_transform(category)

# Creating a DataFrame from the encoded categories with appropriate co
category_columns = enc.get_feature_names_out(input_features = category
category = pd.DataFrame(Cat_new, columns = category_columns)

# Selecting all Numerical Features
num = data.select_dtypes(exclude=["object"])

# Merging Categorical Feature and Numerical Feature
n_data = pd.concat([num, category], axis="columns")

# Replacing null values with mean values
n_data.fillna(n_data.mean(), inplace=True)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning:

`sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
```

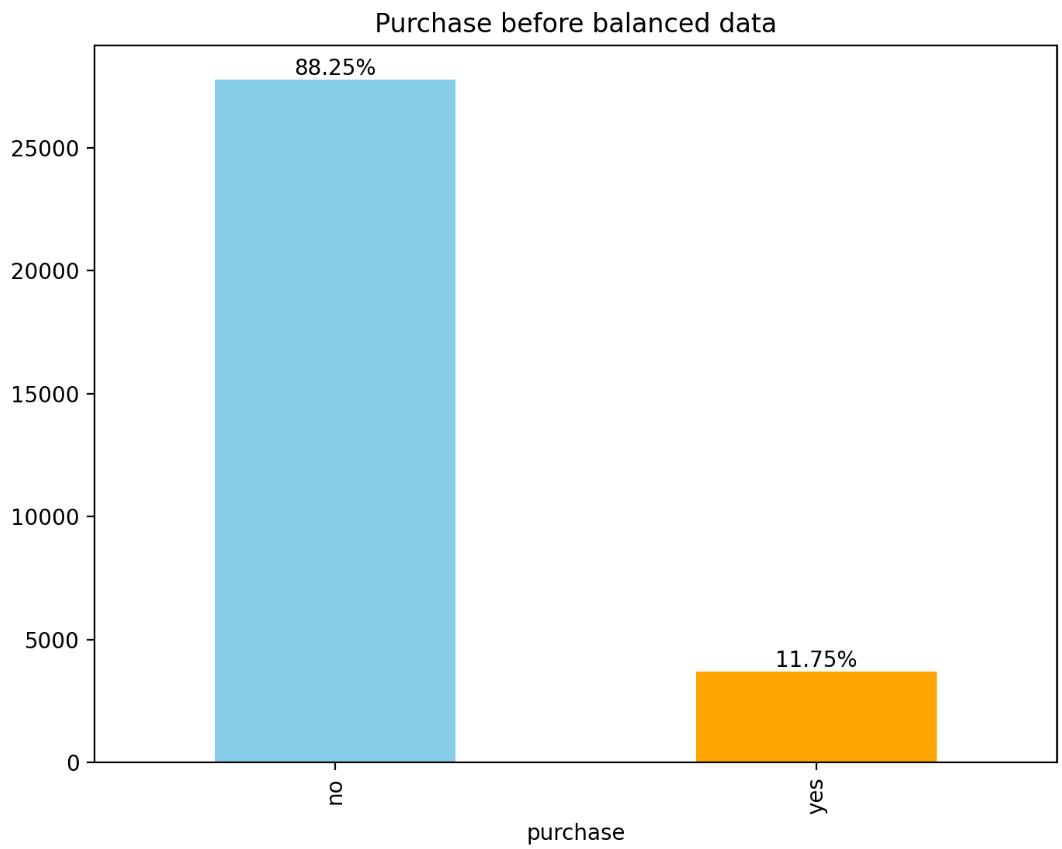
In [34]:
```
n_data
```

Out[34]:

	purchase	day	duration	age	accountBalance	numberOfContacts	numberOfContactsLa
0	0	27	166	30	-202	2	
1	0	26	183	42	2463	2	
2	0	5	227	26	2158	1	
3	0	2	31	34	75	3	
4	0	15	1231	48	559	2	
...	...	...	...	...	...	...	
31475	1	30	1628	58	3399	2	
31476	0	21	173	40	858	1	
31477	0	17	422	51	1414	3	
31478	0	29	69	30	1	21	
31479	0	2	171	50	8	2	

31480 rows × 49 columns

In [35]:
```
counts = df['purchase'].value_counts()

# Plotting the bar chart
counts.plot(kind='bar', figsize=(8, 6), color=['skyblue', 'orange'], t

total = len(df['purchase'])
for index, value in enumerate(counts):
    percentage = f'{(value / total) * 100:.2f}%'
    plt.text(index, value, percentage, ha='center', va='bottom')

plt.show()
```



In [36]:
```
X=n_data.drop(['purchase'], axis=1)
y=n_data['purchase']
```

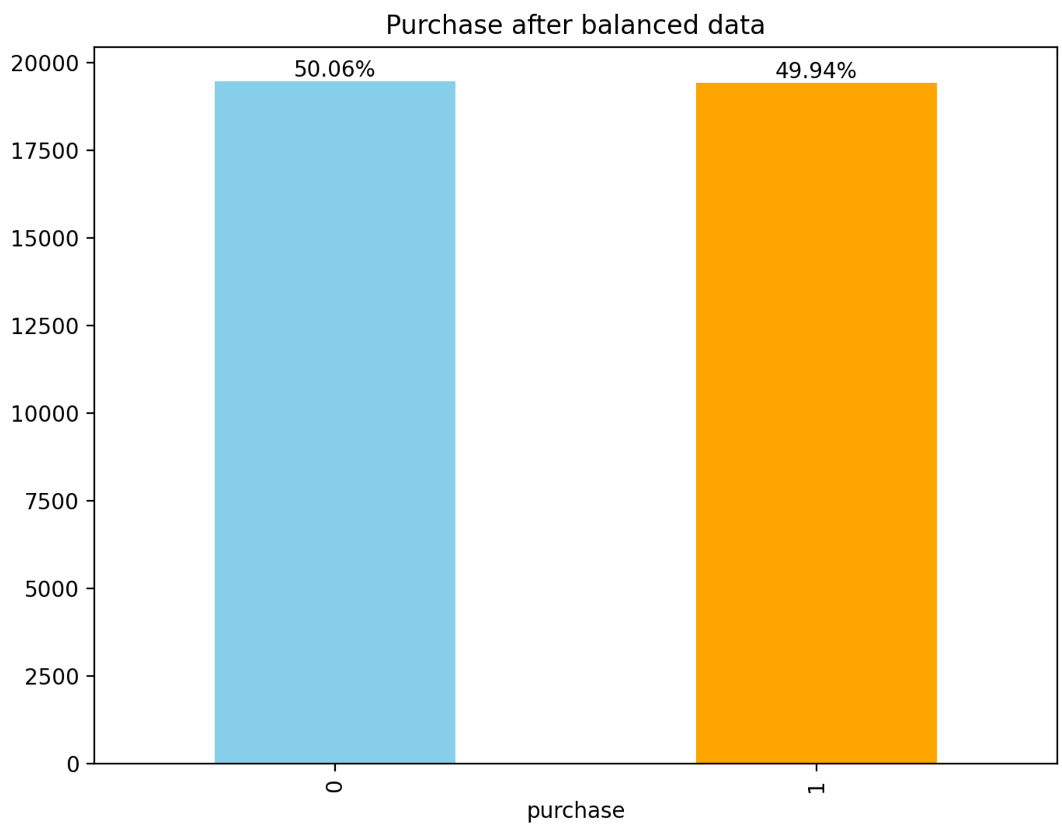
In [37]:

```
# Making the data balanced
ros = RandomOverSampler(sampling_strategy=1)
X_res, y_res = ros.fit_resample(X,y)

# We are spliting data to trian 70% and test 30%
x_train, x_test, y_train, y_test = train_test_split(X_res, y_res, test
```

In [38]:

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(x_train)
X_test_scaled = scaler.transform(x_test)
```

In [39]:
```
t = pd.DataFrame(y_train, columns=['purchase'])

# Calculating the value counts of the 'Purchase' column
counts = t['purchase'].value_counts()

# Plotting the bar chart
counts.plot(kind='bar', figsize=(8, 6), color=['skyblue', 'orange'], t

total = len(t['purchase'])
for index, value in enumerate(counts):
    percentage = f'{(value / total) * 100:.2f}%'
    plt.text(index, value, percentage, ha='center', va='bottom')

plt.show()
```



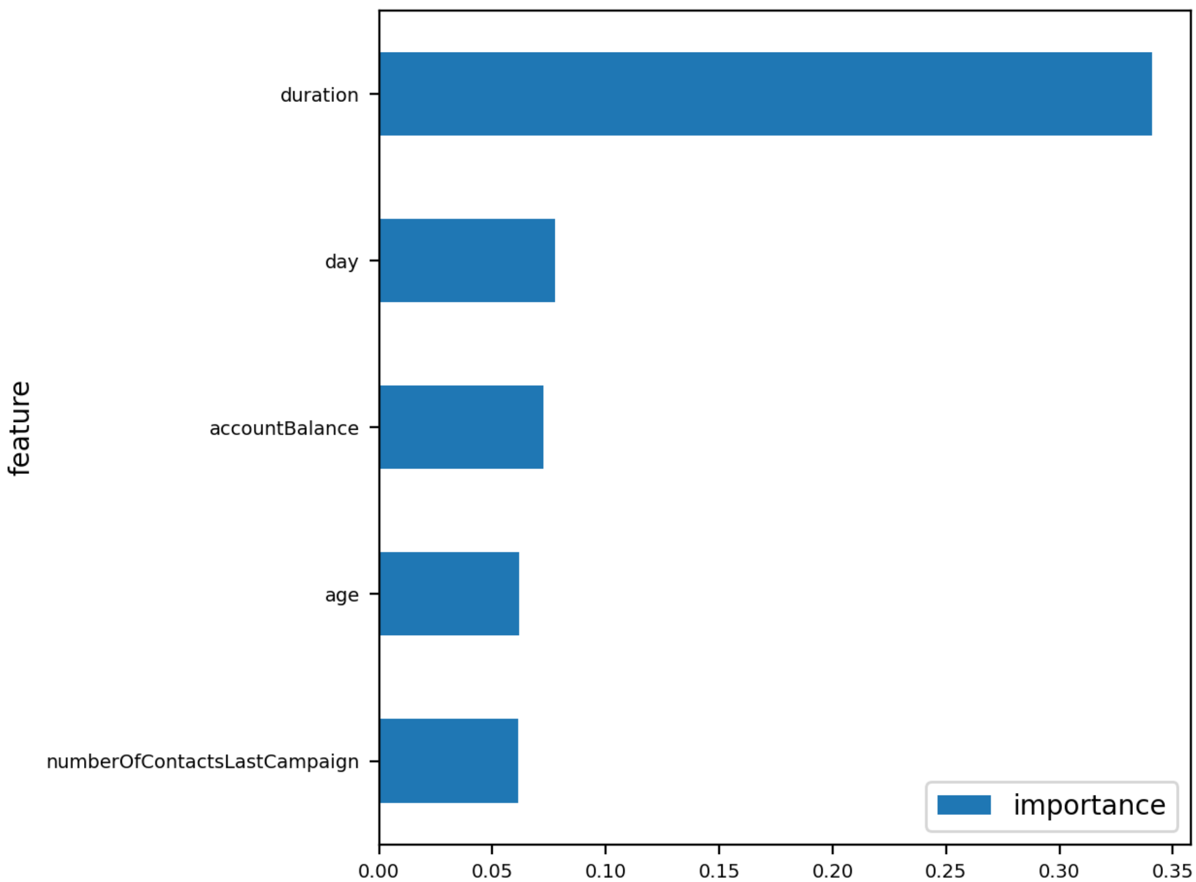
Now the data is balanced

In [40]:
```
X_tree=n_data.drop(['purchase'], axis=1)
y_tree=n_data['purchase']

# Making the data balanced
ros_tree = RandomOverSampler(sampling_strategy=1)
X_res_tree, y_res_tree = ros.fit_resample(X_tree,y_tree)

# We are spliting data to trian 70% and test 30%
x_train_tree, x_test_tree, y_train_tree, y_test_tree = train_test_spli
```

In [41]:
```
from sklearn.tree import DecisionTreeClassifier

dtree_model = DecisionTreeClassifier(random_state=42)

# Fitting the classifier to the training data
dtree_model.fit(x_train_tree, y_train_tree)

# Predicting the labels of the test set
y_pred_tree = dtree_model.predict(x_test_tree)

print("Accuracy for Decision Tree Model: ", accuracy_score(y_test_tree
```

```
Accuracy for Decision Tree Model:  0.9558435325173986
```

In [42]:
```
# Extract importance values for each feature (column of X)
importances = dtree_model.feature_importances_

# create a dataframe to store the values and their labels
df_feature = pd.DataFrame({'feature': x_train_tree.columns, 'importanc

# sort dataframe by descending order, showing the most important featu
df_feature = df_feature.sort_values('importance', ascending = True)

df_feature.set_index('feature', inplace=True)

# Plot the importance of each feature with features on the y-axis
ax = df_feature.tail(5).plot(kind='barh', fontsize=7)

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



In [43]:
```
X
```

Out[43]:

	day	duration	age	accountBalance	numberOfContacts	numberOfContactsLastCampaig
0	27	166	30	-202	2	
1	26	183	42	2463	2	
2	5	227	26	2158	1	
3	2	31	34	75	3	
4	15	1231	48	559	2	
...	...	...	...	...	...	.
31475	30	1628	58	3399	2	
31476	21	173	40	858	1	
31477	17	422	51	1414	3	
31478	29	69	30	1	21	
31479	2	171	50	8	2	

31480 rows × 48 columns

In [44]:
```
df_feature
```

Out[44]:

	importance
feature	
creditFailure_yes	0.000428
creditFailure_no	0.000695
job_retired	0.001259
job_unknown	0.001479
month_dec	0.002648
job_houseWife	0.002771
job_selfEmployed	0.003005
gender_male	0.003171
education_highSchool	0.003244
credit_yes	0.003304
job_unemployed	0.003662
contactType_landline	0.003806
education_unknown	0.004089

maritalStatus_divorced	0.004140
education_secondarySchool	0.004316
maritalStatus_single	0.004566
job_entrepreneur	0.004579
job_services	0.004615
gender_female	0.004767
job_administrative	0.005114
maritalStatus_married	0.005148
job_student	0.005433
education_uniGraduated	0.005451
month_sep	0.006044
job_manager	0.006398
job_worker	0.006594
job_technical	0.007013
month_jul	0.007616
month_jan	0.008555
credit_no	0.008712
month_jun	0.008943
month_aug	0.009128
contactType_cellPhone	0.009313
month_nov	0.009874
month_may	0.012689
month_feb	0.014292
house_no	0.017811
month_oct	0.018223
month_apr	0.020791
month_mar	0.023405
numberOfContacts	0.025882
house_yes	0.033852
contactType_unknown	0.048701
numberOfContactsLastCampaign	0.061489

age	0.061781
accountBalance	0.072691
day	0.077608
duration	0.340905

In [45]:
```
rfc=RandomForestClassifier(random_state=42)
rfc.fit(X_train_scaled,y_train)

pred=rfc.predict(X_test_scaled)

print("Accuracy for Random Forest Classifier data: ",rfc.score(X_test_
```

```
Accuracy for Random Forest Classifier data:  0.9639428845692345
```

In [46]:
```
conf_mat = confusion_matrix(y_test, pred)

# Generate the classification report
class_rep = classification_report(y_test, pred)

# Print the confusion matrix and classification report
print("Confusion Matrix:")
print(conf_mat)

print("\nClassification Report:")
print(class_rep)
```

```
Confusion Matrix:
[[7741  569]
 [  32 8326]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.93      0.96      8310
           1       0.94      1.00      0.97      8358

    accuracy                           0.96     16668
   macro avg       0.97      0.96      0.96     16668
weighted avg       0.97      0.96      0.96     16668
```

In [47]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Feature Scaling: StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Logistic Regression model
log_reg_model = LogisticRegression()

log_reg_model.fit(X_train_scaled, y_train)

y_pred = log_reg_model.predict(X_test_scaled)

# Calculating the accuracy of the logistic regression model
accuracy = log_reg_model.score(X_test_scaled, y_test)

# Printing the results
print("Accuracy for Logistic Regression: ", accuracy)
```

```
Accuracy for Logistic Regression:  0.8982422702244811
```

In [48]:
```
# Generating the confusion matrix and classification report
conf_mat = confusion_matrix(y_test, y_pred)
class_rep = classification_report(y_test, y_pred)

print("Confusion Matrix:")
print(conf_mat)
print("\nClassification Report:")
print(class_rep)
```

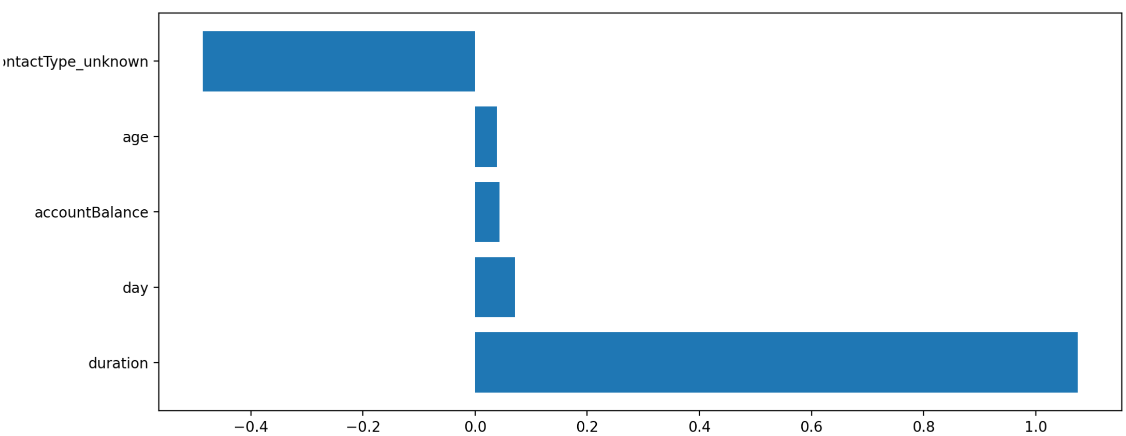
```
Confusion Matrix:
[[8157  194]
 [ 767  326]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.98      0.94      8351
           1       0.63      0.30      0.40      1093

    accuracy                           0.90      9444
   macro avg       0.77      0.64      0.67      9444
weighted avg       0.88      0.90      0.88      9444
```

In [49]:
```
coef=log_reg_model.coef_[0]
df_coef=pd.DataFrame({"Coeffcient":coef},index=X_train.columns)
coef=np.sort(coef)
coef[::-1][:5]
df_coef.sort_values(by="Coeffcient", ascending=False)
top_5_coef=df_coef.loc[["duration", "day", "accountBalance", "age","co
top_5_coef=top_5_coef.sort_values(by="Coeffcient", ascending=False)
top_5_coef
```

Out[49]:

	Coeffcient
duration	1.075500
day	0.071372
accountBalance	0.043125
age	0.038699
contactType_unknown	-0.486155

In [50]:
```
plt.figure(figsize=(12, 5))
plt.barh(top_5_coef.index,top_5_coef["Coeffcient"])
plt.show()
```



In [51]:

```
# import statspackage
import statsmodels.api as sm

# note: statsmodels is missing the constant term in the sigmoid,
# so we need to add it back...
X_train_scaled_labeled=pd.DataFrame(X_train_scaled,columns=X_train.col
X_test_scaled_labeled=pd.DataFrame(X_test_scaled,columns=X_train.colum

X_train_scaled_with_constant = sm.add_constant(X_train_scaled_labeled)
X_test_scaled_with_constant = sm.add_constant(X_test_scaled_labeled)

logit_reg = sm.Logit(y_train, X_train_scaled_with_constant).fit()
print(logit_reg.params[["duration", "day", "accountBalance", "age","co


summary_table = logit_reg.summary()

summary_df = pd.DataFrame(summary_table.tables[1].data[1:], columns=su
summary_df.rename(columns={'''''': 'features'}, inplace=True)
specific_features_summary=summary_df[summary_df["features"].isin(["dur
# or summary_df.iloc[[1,2,3,4,6],:]


specific_features_summary.sort_values(by="coef", ascending=False).rese
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.251289
         Iterations: 35
duration               1.076307
day                    0.071771
accountBalance         0.043111
age                    0.038755
contactType_unknown   -0.487028
dtype: float64

/usr/local/lib/python3.10/dist-packages/statsmodels/base/model.py:607
: ConvergenceWarning:

Maximum Likelihood optimization failed to converge. Check mle_retvals
```

Out[51]:

	features	coef	std err	z	P>|z|	[0.025	0.975]
0	contactType_unknown	-0.4870	7.09e+05	-6.87e-07	1.000	-1.39e+06	1.39e+06
1	duration	1.0763	0.023	46.041	0.000	1.030	1.122
2	day	0.0718	0.029	2.487	0.013	0.015	0.128
3	accountBalance	0.0431	0.022	1.983	0.047	0.001	0.086

4	age	0.0388	0.033	1.192	0.233	-0.025	0.102

In [52]:
```
import pandas as pd
import math

# Assuming specific_features_summary is your DataFrame
specific_features_summary['coef'] = pd.to_numeric(specific_features_su

# Now you can apply math.exp
specific_features_summary['exp_coef'] = specific_features_summary['coe

specific_features_summary.reset_index(drop=True)
```

```
<ipython-input-52-b3097db5f217>:5: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)

<ipython-input-52-b3097db5f217>:8: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
(https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.htm
l#returning-a-view-versus-a-copy)
```

Out[52]:

	features	coef	std err	z	P>|z|	[0.025	0.975]	exp_coef
0	day	0.0718	0.029	2.487	0.013	0.015	0.128	1.074440
1	duration	1.0763	0.023	46.041	0.000	1.030	1.122	2.933804
2	age	0.0388	0.033	1.192	0.233	-0.025	0.102	1.039563
3	accountBalance	0.0431	0.022	1.983	0.047	0.001	0.086	1.044042
4	contactType_unknown	-0.4870	7.09e+05	-6.87e-07	1.000	-1.39e+06	1.39e+06	0.614467

In [53]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn_model = KNeighborsClassifier(n_neighbors=10)

knn_model.fit(X_train_scaled, y_train)

y_pred = knn_model.predict(X_test_scaled)

# Calculating the accuracy of the KNN model on the scaled test data
accuracy = knn_model.score(X_test_scaled, y_test)

# Generating the confusion matrix and classification report
conf_mat = confusion_matrix(y_test, y_pred)
class_rep = classification_report(y_test, y_pred)

print("Accuracy for KNN Classifier: ", accuracy)
```

```
Accuracy for KNN Classifier:  0.8846886912325286
```

In [54]:
```
# Generating the confusion matrix and classification report
conf_mat = confusion_matrix(y_test, y_pred)
class_rep = classification_report(y_test, y_pred)

print("Confusion Matrix:")
print(conf_mat)

print("\nClassification Report:")
print(class_rep)
```

```
Confusion Matrix:
[[8205  146]
 [ 943  150]]

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      8351
           1       0.51      0.14      0.22      1093

    accuracy                           0.88      9444
   macro avg       0.70      0.56      0.58      9444
weighted avg       0.85      0.88      0.85      9444
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Build the neural network
model = Sequential()
model.add(Dense(10, input_dim=X_train_scaled.shape[1], activation='rel
model.add(Dense(5, activation='relu')) # Hidden layer
model.add(Dense(1, activation='sigmoid')) # Output layer for binary cl

# Compiling the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['

# Training the model
model.fit(X_train_scaled, y_train, epochs=100, batch_size=10)

_, accuracy = model.evaluate(X_test_scaled, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Epoch 1/100
2204/2204 [==============================] - 5s 2ms/step - loss: 0.27
98 - accuracy: 0.8831
Epoch 2/100
2204/2204 [==============================] - 4s 2ms/step - loss: 0.23
16 - accuracy: 0.8959
Epoch 3/100
2204/2204 [==============================] - 6s 3ms/step - loss: 0.22
50 - accuracy: 0.8983
Epoch 4/100
2204/2204 [==============================] - 5s 2ms/step - loss: 0.22
09 - accuracy: 0.8985
Epoch 5/100
2204/2204 [==============================] - 4s 2ms/step - loss: 0.21
72 - accuracy: 0.9012
Epoch 6/100
2204/2204 [==============================] - 6s 3ms/step - loss: 0.21
48 - accuracy: 0.9012
Epoch 7/100
```

In [ ]:
```
# Assuming y_test are true labels and y_pred are your predictions
y_pred = model.predict(X_test_scaled)

# For binary classification, a threshold of 0.5 is settled
y_pred_binary = (y_pred > 0.5).astype("int32")

conf_matrix = confusion_matrix(y_test, y_pred_binary)

class_rep = classification_report(y_test, y_pred_binary)

print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_rep)
```

In [ ]:
```
# Accuracy Score Data Frame

Acc_index = np.array([0.9659827213822895, 0.8982422702244811, 0.884688
Acc_index = np.round(Acc_index, 3)
Acc_column = ["Random Forest", "Logistic Regression", "KNN", "Neural N
Acc_df = pd.DataFrame(Acc_index.reshape(1,4), columns=Acc_column, inde
Acc_df
```

Testing it using the test dataset

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from plotnine import ggplot, aes, geom_boxplot, labs, theme, element_t
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
df_test = pd.read_csv('/content/drive/My Drive/Colab Data/test.csv')
df_test.info()
```

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

dft=df_test.copy()
dft.columns = ('id', 'purchase', 'day', 'month', 'duration', 'contactI
       'gender', 'job', 'maritalStatus', 'education', 'creditFailure',
       'accountBalance', 'house', 'credit', 'contactType', 'numberOfCo
       'daySinceLastCampaign', 'numberOfContactsLastCampaign',
       'lastCampaignResult')
```

In [ ]:
```
dft['daySinceLastCampaign'].fillna(-1, inplace=True)
```

In [ ]:
```
dft_id=dft["id"]
dft_job_ab_st=dft[["job", 'accountBalance', "duration", "education"]]
dft=dft.drop(["daySinceLastCampaign","lastCampaignResult","id","contac
```

In [ ]:
```
dft=dft.drop(["purchase"], axis=1)
```

In [ ]:
```
# Selecting all Categorical Features
category = dft.select_dtypes(include=["object"])

enc = OneHotEncoder(sparse=False)

# Applying OneHotEncoder to the categorical data
Cat_new = enc.fit_transform(category)

# Creating a DataFrame from the encoded categories with appropriate co
category_columns = enc.get_feature_names_out(input_features = category
category = pd.DataFrame(Cat_new, columns = category_columns)

# Selecting all Numerical Features
num = dft.select_dtypes(exclude=["object"])

# Merging Categorical Feature and Numerical Feature
dft2 = pd.concat([num, category], axis="columns")

# Replacing null values with mean values
dft2.fillna(dft2.mean(), inplace=True)
```

In [ ]:
```
X_ts=dft2
scaler = StandardScaler()

X_ts_scaled = scaler.fit_transform(X_ts)
y_pred = rfc.predict(X_ts_scaled)

# Comparing predictions to the actual values
pred = pd.DataFrame({'Predicted': y_pred},index=dft_id)
pred
```

In [ ]:
```
test_ids = [432184585, 432167969, 432146206]
specific_pred = pred[pred.index.isin(test_ids)]
# Step 4: Display the result
specific_pred
```

In [ ]:
```
Probs = rfc.predict_proba(X_ts_scaled)
test_predict = Probs[:,1]
df_prob=pd.DataFrame(columns=["Test ID", "Expected", "Job", "Account E
df_prob["Test ID"]=dft_id.values
df_prob["Expected"]=test_predict
df_prob["Job"]=dft_job_ab_st["job"]
df_prob["Account Balance"]=dft_job_ab_st["accountBalance"]
df_prob["Education"]=dft_job_ab_st["education"]
df_prob["Duration"]=dft_job_ab_st["duration"]
df_prob
```

In [ ]:
```
test_ids = [432184585, 432167969, 432146206]
specific_pred_with_features = df_prob[df_prob["Test ID"].isin(test_ids
specific_pred_with_features
```