Projekt – sieci neuronowe	Data złożenia projektu:14.05.2021
Numer grupy projektowej: 02	Imię i nazwisko I: Karol Matoga Imię i nazwisko II: Dariusz Nowak

Predict the probability of an candidate looking for a new job

1. Opis problemu i danych

Firmy poszukujące pracowników, chcą wiedzieć czy ich kandydaci do zatrudnienia, którzy ukończyli pewne kursy wymagane do rekrutacji, rzeczywiście chcą pracować w ich firmie. W projekcie wykorzystano zbiór danych “HR Analytics: Job Change of Data Scientists”, zawierający 19158 rekordów, z których każdy zawiera 14 następujących zmiennych:

- Ilościowych:
    - city_development_index : Wskaźnik rozwoju miasta
    - company_size: Liczba pracowników w aktualnej firmie
    - last*new*job: Róznica w latach pomiędzy poprzednią, a aktualną pracą
    - training_hours: Ukończone godziny ćwiczeń
- Jakościowych:
    - enrollee_id : unikalne ID dla kandydata
    - city: kod miasta
    - gender: płeć kandydata
    - relevent_experience: Czy kandydat ma doświadczenie w wymaganej dziedzinie.
    - enrolled_university: Typ odbytych studiów
    - education_level: Poziom wykształcenia
    - experience: Doświadczenie zawodowe w latach
    - company_type: Typ obecnego pracodawcy
    - target: 0 – Nie chce zmienić pracy, 1 – Chce zmienić pracę

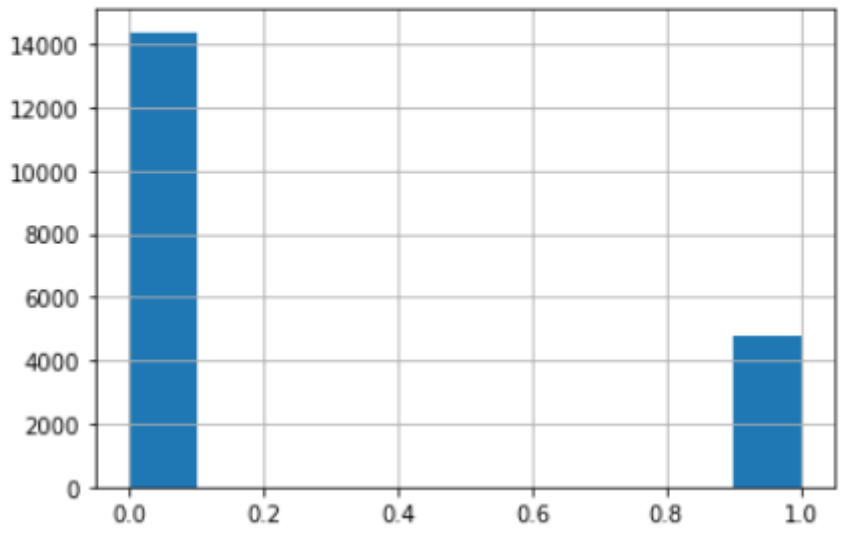
```
df.describe()
```

	enrollee_id	city_development_index	training_hours	target
count	19158.000000	19158.000000	19158.000000	19158.000000
mean	16875.358179	0.828848	65.366896	0.249348
std	9616.292592	0.123362	60.058462	0.432647
min	1.000000	0.448000	1.000000	0.000000
25%	8554.250000	0.740000	23.000000	0.000000
50%	16982.500000	0.903000	47.000000	0.000000
75%	25169.750000	0.920000	88.000000	0.000000
max	33380.000000	0.949000	336.000000	1.000000

Podstawowe statystyki

```
df["target"].hist()
```

```
<AxesSubplot:>
```



Histogram zmiennej target - zmiennej wyjściowej

Celem projektu było rozwiązanie problemu określania prawdopodobieństwa, że dana osoba chce zmienić pracę. Rozwiązywany problem dotyczył klasyfikacji. Na wyjściu możemy uzyskać dwie możliwości: dana osoba nie chce zmienić pracy, albo chce zmienić pracę.

2. Obróbka danych

Zbiór danych nie posiadał wartości dla niektórych rekordów w odpowiednich kolumnach. Dodatkowo posiadał dane kategoryczne, które do poprawnego rozwiązania należało zamienić na wartości numeryczne albo wyrzucić.

In [12]:
```
def gender_to_numeric(x):
    if x=='Female': return 2
    if x=='Male':   return 1
    if x=='Other':   return 0

def rel_experience(x):
    if x=='Has relevent experience': return 1
    if x=='No relevent experience':   return 0

def enrollment(x):
    if x=='no_enrollment'   : return 0
    if x=='Full time course':   return 1
    if x=='Part time course':   return 2

def edu_level(x):
    if x=='Graduate'       :   return 0
    if x=='Masters'        :   return 1
    if x=='High School'    :   return 2
    if x=='Phd'            :   return 3
    if x=='Primary School' :   return 4

def major(x):
    if x=='STEM'                   :   return 0
    if x=='Business Degree'        :   return 1
    if x=='Arts'                   :   return 2
    if x=='Humanities'             :   return 3
    if x=='No Major'               :   return 4
    if x=='Other'                  :   return 5
```

Ustalenie zmiennych numerycznych dla zmiennych kategorycznych

In [13]:
```
df['gender'] = df['gender'].apply(gender_to_numeric)
df['relevent_experience'] = df['relevent_experience'].apply(rel_experience)
df['enrolled_university'] = df['enrolled_university'].apply(enrollment)
df['education_level'] = df['education_level'].apply(edu_level)
df['major_discipline'] = df['major_discipline'].apply(major)
df['experience'] = df['experience'].apply(experience)
df['company_type'] = df['company_type'].apply(company_t)
df['company_size'] = df['company_size'].apply(company_s)
df['last_new_job'] = df['last_new_job'].apply(last_job)
df['city'] = df['city'].apply(city)

df
```

Aktualna zmiana danych

Out[13]:

	enrollee_id	city	city_development_index	gender	relevent_experience	enrolled_university	education_level	major_discipline	experience	company_size
0	8949	0	0.920	1.0	1	0.0	0.0	0.0	21.0	NaN
1	29725	1	0.776	1.0	0	0.0	0.0	0.0	15.0	5.0
2	11561	2	0.624	NaN	0	1.0	0.0	0.0	5.0	NaN
3	33241	3	0.789	NaN	0	NaN	0.0	1.0	0.0	NaN
4	666	4	0.767	1.0	1	0.0	1.0	0.0	21.0	5.0
...	...	...	...	...	...	...	...	...	...	...
19153	7386	35	0.878	1.0	0	0.0	0.0	3.0	14.0	NaN
19154	31398	0	0.920	1.0	1	0.0	0.0	0.0	14.0	NaN
19155	24576	0	0.920	1.0	1	0.0	0.0	0.0	21.0	5.0
19156	5756	33	0.802	1.0	1	0.0	2.0	NaN	0.0	6.0
19157	23834	13	0.855	NaN	0	0.0	4.0	NaN	2.0	NaN

Zmienione dane bez zamiany wartości null

In [15]:
```
df['gender'] = df['gender'].fillna((df['gender'].mean()))
df['enrolled_university'] = df['enrolled_university'].fillna((df['enrolled_university'].mean()))
df['major_discipline'] = df['major_discipline'].fillna((df['major_discipline'].mean()))
df['company_size'] = df['company_size'].fillna((df['company_size'].mean()))
df['company_type'] = df['company_type'].fillna((df['company_type'].mean()))
df['company_type'] = df['company_type'].fillna((df['company_type'].mean()))

df
```

Zamiana wartości null na średnie

Out[15]:

ee_id	city	city_development_index	gender	relevent_experience	enrolled_university	education_level	major_discipline	experience	company_size	company_typ
8949	0	0.920	1.000000	1	0.000000	0.0	0.000000	21.0	3.27466	0.6909€
29725	1	0.776	1.000000	0	0.000000	0.0	0.000000	15.0	5.00000	0.00000
11561	2	0.624	1.071468	0	1.000000	0.0	0.000000	5.0	3.27466	0.6909€
33241	3	0.789	1.071468	0	0.327775	0.0	1.000000	0.0	3.27466	0.00000
666	4	0.767	1.000000	1	0.000000	1.0	0.000000	21.0	5.00000	1.00000
...	...	...	...	...	...	...	...	...	...	...
7386	35	0.878	1.000000	0	0.000000	0.0	3.000000	14.0	3.27466	0.6909€
31398	0	0.920	1.000000	1	0.000000	0.0	0.000000	14.0	3.27466	0.6909€
24576	0	0.920	1.000000	1	0.000000	0.0	0.000000	21.0	5.00000	0.00000
5756	33	0.802	1.000000	1	0.000000	2.0	0.344876	0.0	6.00000	0.00000
23834	13	0.855	1.071468	0	0.000000	4.0	0.344876	2.0	3.27466	0.6909€

Po wykonaniu owych operacji posiadano dalej wartości null w tabeli.

Out[20]:

	Number of null values	Percentage null values
enrollee_id	0	0.00
city	0	0.00
city_development_index	0	0.00
gender	0	0.00
relevent_experience	0	0.00
enrolled_university	0	0.00
education_level	460	2.40
major_discipline	0	0.00
experience	65	0.34
company_size	0	0.00
company_type	0	0.00
last_new_job	423	2.21
training_hours	0	0.00
target	0	0.00

Wyrzucono rekordy, które nie posiadały wartości

```
In [21]: df = df.dropna(axis=0)
```

```
Out[22]: enrollee_id              False
         city                     False
         city_development_index   False
         gender                   False
         relevent_experience      False
         enrolled_university      False
         education_level          False
         major_discipline         False
         experience               False
         company_size             False
         company_type             False
         last_new_job             False
         training_hours           False
         target                   False
         dtype: bool
```

Brak wartości null

Wyznaczono X (wejście do neuronu)

```
In [28]: X = df.drop("target", axis=1)
```

oraz y (wyjście neuronu)

```
In [30]: y = df["target"]
```

Podział na zbiór uczący oraz trenujący (walidacja)

```
In [48]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
```

Ustawiliśmy test_size na 0.15 dlatego aby kształt histogramu dla danych trenujących oraz testujących były zbliżone.

Posiadano do dyspozycji plik “aug_test.csv”, który obrobiono tak samo jak “aug_train.csv”. “test_size” obliczono następująco (df dla trenowania):

(ilość_rekordów_dla_df_test)/(ilość_rekordów_dla_df)

“random_state” to seed dla generatora liczb pseudolosowych (sklearn) aby generowane zestawy danych dla test oraz train zawsze były identyczne dla kolejnych wywołań.

3. Opis zastosowanych sieci neuronowych

Architektura nr 1:

- typ: gęsta sieć neuronowa
- ilość epok uczenia: 500
- sposób uczenia sieci: funkcja aktywacji - relu
- framework: tensorflow nakładka keras
- ilość danych wejściowych: 3
- warstwa nr 1 (ukryta): 500 neuronów, relu
- warstwa nr 2 (ukryta): 500 neuronów, relu
- warstwa nr 3 (ukryta): 250 neuronów, relu
- warstwa nr 4 (ukryta): 250 neuronów, relu
- warstwa nr 5 (ukryta): 125 neuronów, relu
- warstwa nr 6 (wyjście): 2 neurony (2 klasy, wartości 0 oraz 1, multi-klasyfikacja), softmax
- funkcja stary: “sparse_categorical_crossentropy”
- optymalizator: “adam”
- metryka: “accuracy”

In [57]:
```
model = Sequential([
    Input(shape=X_train.shape[1]),
    Dense(500, activation="relu"),
    Dense(500, activation="relu"),
    Dense(250, activation="relu"),
    Dense(250, activation="relu"),
    Dense(125, activation="relu"),
    Dense(2, activation="softmax")
])
```

In [58]:
```
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer="adam",
    metrics=["accuracy"])
```

Architektura nr 1 z uściślonymi danymi (StandardScaler from sklearn.preprocessing).

Architektura nr 2 (tensorflow jest tak zbudowany, że szybciej działa jak ma na wyjściu 1 neuron nawet jeśli mamy 2 klasy (0 oraz 1)):

- typ: gęsta sieć neuronowa
- ilość epok uczenia: 100
- sposób uczenia sieci: funkcja aktywacji - relu
- framework: tensorflow nakładka keras
- ilość danych wejściowych: 3
- warstwa nr 1 (ukryta): 500 neuronów, relu
- warstwa nr 2 (ukryta): 500 neuronów, relu
- warstwa nr 3 (ukryta): 250 neuronów, relu
- warstwa nr 4 (ukryta): 250 neuronów, relu
- warstwa nr 5 (ukryta): 125 neuronów, relu
- warstwa nr 6 (wyjście): 1 neuron (1 klasa, wartości 0 oraz 1, klasyfikacja), sigmoid
- funkcja stary: “binary_crossentropy”
- optymalizator: “adam”
- metryka: “binary_accuracy”

In [86]:
```
model3 = Sequential([
    Input(shape=X_train.shape[1]),
    Dense(500, activation="relu"),
    Dense(500, activation="relu"),
    Dense(250, activation="relu"),
    Dense(250, activation="relu"),
    Dense(125, activation="relu"),
    Dense(1, activation="sigmoid")
])
```

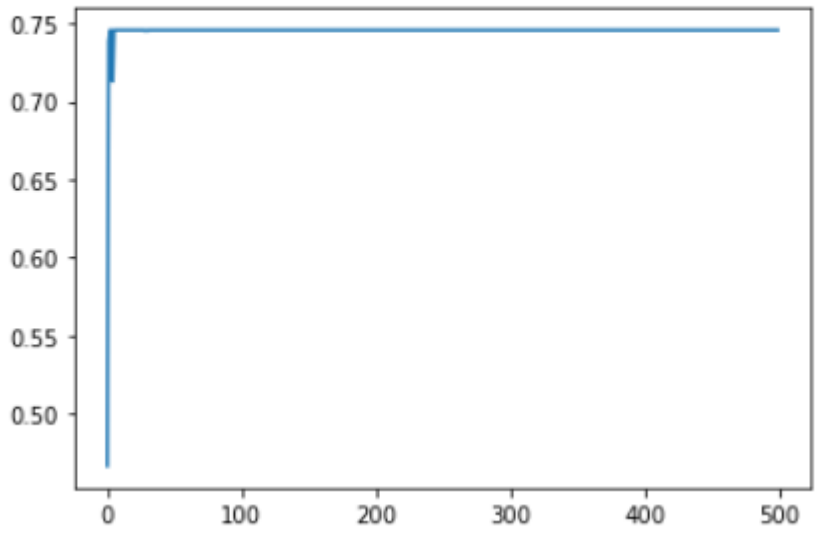
In [87]:
```
model3.compile(
    loss="binary_crossentropy",
    optimizer="adam",
    metrics=["binary_accuracy"])
```

4. Dyskusja wyników oraz wnioski

Zebrane w tabeli wyniki sieci z podziałem na zbiór uczący i testujący. Co najmniej 2 miary, wraz z komentarzem, dlaczego zostały wybrane takie, a nie inne. Określenie (subiektywne), czy wyniki są satysfakcjonujące wraz z uzasadnieniem. Wnioski, dalsze propozycje rozwoju projektu.

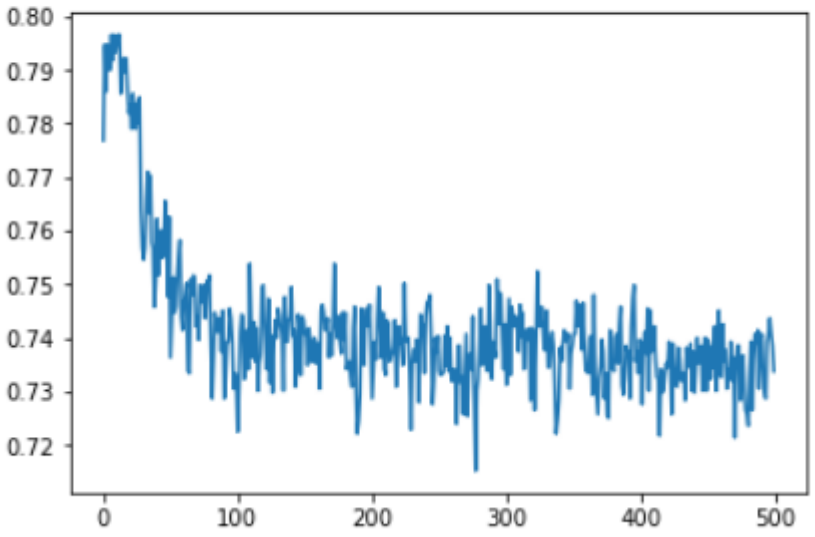
```
In [63]: plt.plot(history.history["val_accuracy"])
```

```
Out[63]: [<matplotlib.lines.Line2D at 0x11a9e268a90>]
```



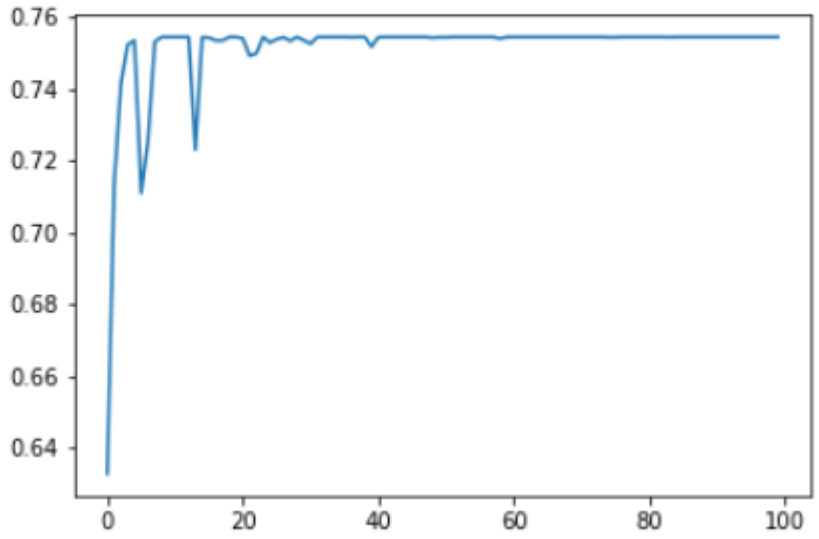

```
In [74]: plt.plot(history2.history["val_accuracy"])
```

```
Out[74]: [<matplotlib.lines.Line2D at 0x11aa08f9490>]
```



In [91]:
```
plt.plot(history3.history["binary_accuracy"])
```

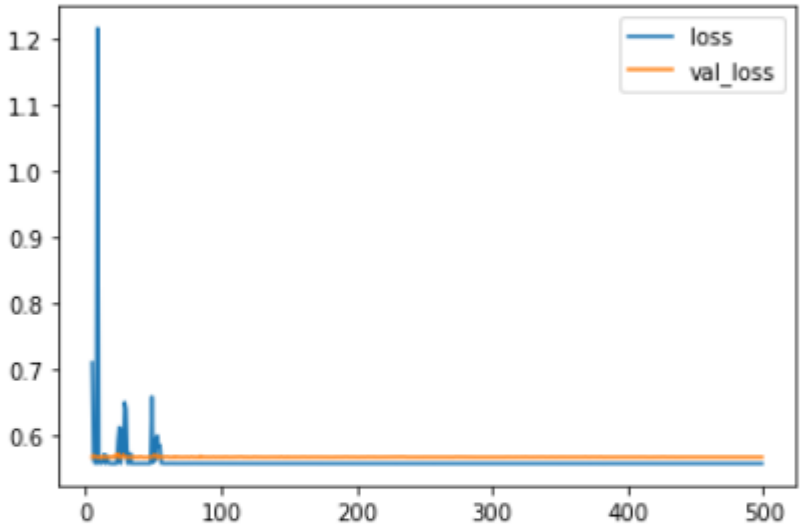
Out[91]:
```
[<matplotlib.lines.Line2D at 0x11aa5230d60>]
```

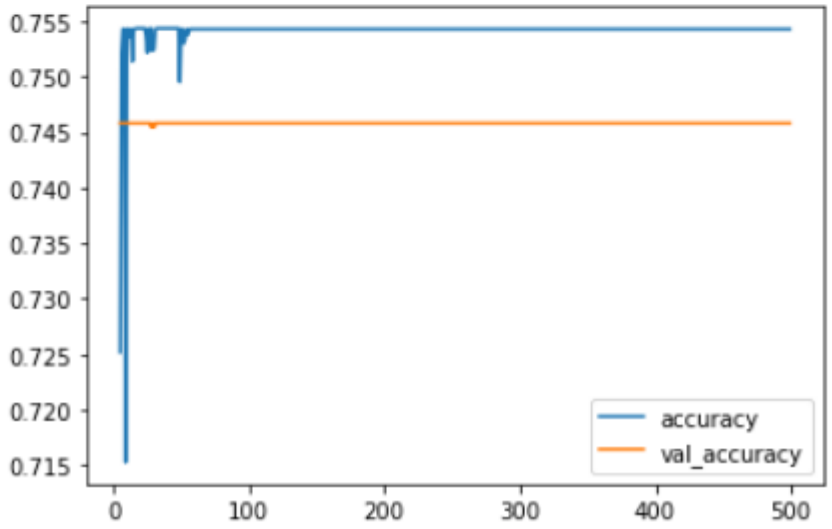


In [92]:
```
history_df = pd.DataFrame(history.history)
history_df.loc[5:, ['loss', 'val_loss']].plot()
history_df.loc[5:, ['accuracy', 'val_accuracy']].plot()

print(("Best Validation Loss: {:0.4f}" +\
      "\nBest Validation Accuracy: {:0.4f}")\
      .format(history_df['val_loss'].min(),
              history_df['val_accuracy'].max()))
```

```
Best Validation Loss: 0.5665
Best Validation Accuracy: 0.7458
```

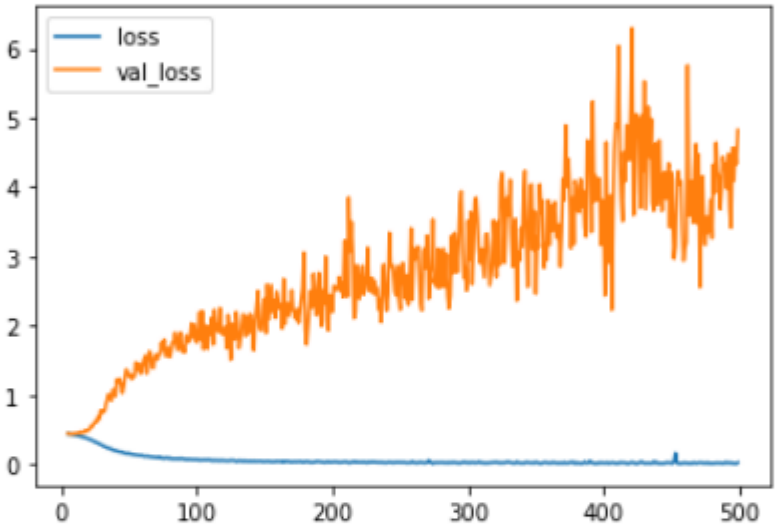


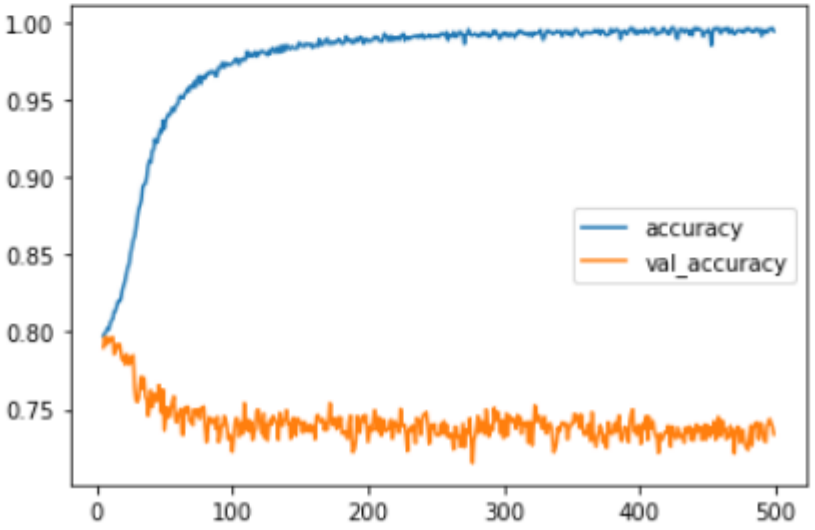


In [93]:
```
history_df = pd.DataFrame(history2.history)
history_df.loc[5:, ['loss', 'val_loss']].plot()
history_df.loc[5:, ['accuracy', 'val_accuracy']].plot()

print(("Best Validation Loss: {:0.4f}" +\
      "\nBest Validation Accuracy: {:0.4f}")\
      .format(history_df['val_loss'].min(),
              history_df['val_accuracy'].max()))
```

```
Best Validation Loss: 0.4346
Best Validation Accuracy: 0.7965
```

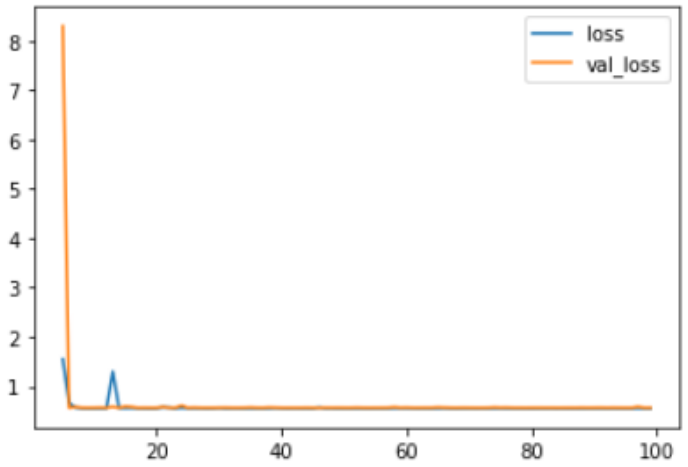


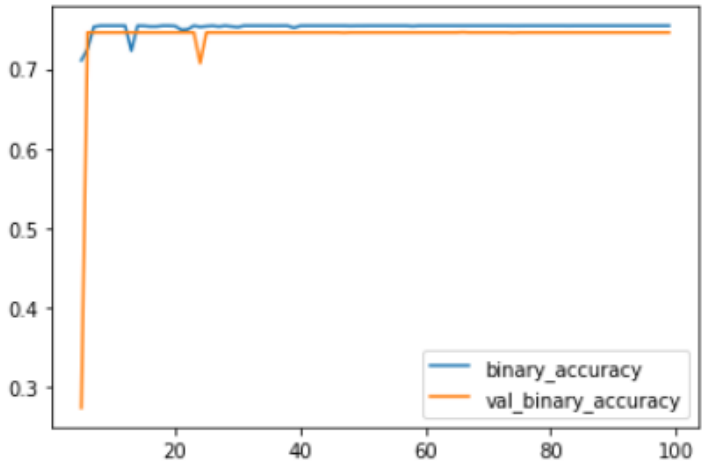


In [94]:
```
history_df = pd.DataFrame(history3.history)
history_df.loc[5:, ['loss', 'val_loss']].plot()
history_df.loc[5:, ['binary_accuracy', 'val_binary_accuracy']].plot()

print(("Best Validation Loss: {:0.4f}" +\
      "\nBest Validation Accuracy: {:0.4f}")\
      .format(history_df['val_loss'].min(),
              history_df['val_binary_accuracy'].max()))
```

```
Best Validation Loss: 0.5657
Best Validation Accuracy: 0.7462
```





architektura	**val loss**	**val accuracy**
architektura nr 1	0.5665	0.7458
architektura nr 1 (z ustandaryzowanymi danymi)	0.4346	0.7965
architektura nr 2	0.5657	0.7462

Wyniki dla dwóch pierwszych architektur można uznać za zadowalające. Z jednej strony dokładność w architekturze nr 1 jest niższa od dokładności w architekturze nr 1 (z ustandaryzowanymi danymi) oraz posiada wyższy wskaźnik validation_loss (“zjada” więcej “noise” danych co jest niepożądane) aczkolwiek ma mniejszą “dziurę” pomiędzy validation-accuracy, a validation-loss w efekcie posiadając mniejszą ilość “noise” danych, danych, które nie służą do tworzenia nowych przewidywań. Można temu zapobiec tworząc “callbacki”. Architektura nr 2 posiada subiektywnie najlepszy rezultat ponieważ dziura miedzy validation-loss, a validation-accuracy jest najmniejsza przy akceptowalnych ich wartościach.

Co można poprawić? Na pewno można przetestować większą liczbę architektur sieci:

- sieci z większą/mniejszą ilością neuronów
- gęstsze/niegęstsze sieci
- różnorakie funkcje aktywacji (poza relu)

Lepsza obróbka danych jeżeli chodzi o zamianę kategorycznych danych na numeryczne oraz pozbycie się wartości null.

Inne typy sieci neuronowych:

- Recurrent Neural Networks (RNN)
- Convolution Neural Networks (CNN)

Zastosowanie “dropoutów” (“wyłączanie” z pracy neuronów przez co sieci może się nauczyć czegoś więcej poprzez zwiększone obciążenie w danym momencie).

Zastosowanie “Batch normalization”, coś w style StandardScaler ale podczas “przerzutów” danych pomiędzy neuronami już w sieci.

No i na końcu zastosowanie innych optymizerów.